MCF5251

Reference Manual

Document Number: MCF5251RM
Rev. 1
01/16/2008

freescale™
semiconductor

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

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

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

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

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

***For Literature Requests Only*:**
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Contents

Chapter 1
MCF5251 Introduction

Chapter 2
Signal Description

Chapter 3
ColdFire Core

Chapter 4
Phase-Locked Loop and Clock Dividers

Chapter 5
Instruction Cache

Chapter 6
Static RAM (SRAM)

Chapter 7
Synchronous DRAM Controller Module

Chapter 8
Bus Operation

Chapter 9
System Integration Module (SIM)

Chapter 10
Chip Select Module

Chapter 11
General Purpose Timer Modules

Chapter 12
Analog to Digital Converter (ADC)

Chapter 13
IDE and Flash Media Interface

Chapter 14
DMA Controller

Chapter 15
UART Modules

Chapter 16
Queued Serial Peripheral Interface (QSPI) Module

Chapter 17
Audio Interface Module (AIM)

Chapter 18
I²C Modules

Chapter 19
Boot ROM

Chapter 20
Background Debug Mode (BDM) Interface

Chapter 21
IEEE 1149.1 Test Access Port (JTAG)

Chapter 22
USB, ATA DMA, and Clock Integration Module

Chapter 23
Advanced Technology Attachment Controller (ATA)

Chapter 24
Universal Serial Bus Interface

Chapter 25
FlexCAN Module

Chapter 26
Real-Time Clock

About This Book

The MCF5251 is designed as a system controller/decoder for compressed audio music players addressing both portable and automotive solutions supporting CD, HDD and USB based systems. The 32-bit ColdFire® core with an enhanced multiply and accumulate (eMAC) unit provides optimum performance and code density for the combination of control code and signal processing required for compressed audio decode, file management, and system control.

Audience

The MCF5251 Reference Manual is intended to provide a design engineer with the necessary data to successfully integrate the MCF5251 into a wide variety of applications. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of software and hardware, and basic details of the ColdFire architecture.

Organization

The MCF5251 Reference Manual is organized into 26 chapters that cover the operation and programming of the MCF5251 device. Summaries of the chapters follow.

Chapter 1, “MCF5251 Introduction”: This chapter provides an overview of the MCF5251 ColdFire® processor and general descriptions of the MCF5251 features and modules.

Chapter 2, “Signal Description”: This chapter describes the MCF5251 input and output signals, organized into functional groups.

Chapter 3, “ColdFire Core”: This chapter provides an overview of the MCF5251 microprocessor core. The chapter describes the CFV2 memory map and register definitions as implemented on the MCF5251. It also includes a full description of exception handling, data formats, an instruction set summary, and a table of instruction timings.

Chapter 4, “Phase-Locked Loop and Clock Dividers”: This chapter provides detailed information about the operation and programming of the clock generation module as well as the recommended circuit settings. It also describes the audio clock generation and the system power states.

Chapter 5, “Instruction Cache”: This chapter describes the physical organization, operation, memory map, and register definitions for the MCF5251 instruction cache.

Chapter 6, “Static RAM (SRAM)”: This chapter describes the SRAM operation, memory map, register definitions, initialization, and SRAM power management.

Chapter 7, “Synchronous DRAM Controller Module”: This chapter discusses the operation, memory map, register definitions, signal and command descriptions, and an interface example for the SDRAM controller.

Chapter 8, “Bus Operation”:This chapter describes bus functionality, the bus control signals, and the bus cycles provided for data-transfer operations. Bus operation is defined for transfers initiated by the MCF5251 as a bus master and for transfers initiated by an alternate bus master. This chapter includes descriptions of the error conditions, bus arbitration, and the reset operation.

Chapter 9, “System Integration Module (SIM)”:This chapter describes the operation, memory map, and register definitions of the System Integration Module (SIM) registers, including the interrupt controller and system-protection functions for the MCF5251. The SIM provides overall control of the internal and external buses and serves as the interface between the ColdFire® core and the internal peripherals or external devices. The SIM also configures the general purpose input/output and enables the CPU HALT instruction.

Chapter 10, “Chip Select Module”:The Chip Select Module provides user-programmable control of the three chip select outputs, two buffer enable outputs and one output-enable signal. This chapter describes the operation, memory map, and register definitions of the chip-select registers, including the chip select address, mask, and control registers.

Chapter 11, “General Purpose Timer Modules”: This chapter describes the configuration and operation of the two general purpose timer modules (Timer0 and Timer1). Also provided are the memory map and register definitions as well as example initialization code.

Chapter 12, “Analog to Digital Converter (ADC)”: This chapter explains the ADC operation, memory map, register definitions, and setup recommendations of external components.

Chapter 13, “IDE and Flash Media Interface”:This chapter describes the operation of the bus interface to IDE and Flash Media, the interface setup, timing and operation are provided as well as commonly used commands.

Chapter 14, “DMA Controller”: This chapter provides the DMA signal descriptions, memory map, register definitions, as well as discussing transfer generation, transfer modes, and the transfer function.

Chapter 15, “UART Modules”: This chapter provides signal descriptions, operation, memory map, register definitions, and initialization sequence of the three UART modules.

Chapter 16, “Queued Serial Peripheral Interface (QSPI) Module”: This chapter describes the operation of the Queued Serial Peripheral interface module and provides its memory map and register definitions. The QSPI module provides a serial peripheral interface with queued transfer capability. It allows users to queue up to 16 transfers at once, eliminating CPU intervention between transfers.

Chapter 17, “Audio Interface Module (AIM)”:This chapter discusses the audio interface structure, memory map, and register definitions, as well as transmit and receive interfaces. The audio interface module provides the necessary input and output features to receive and transmit digital audio signals over serial audio interfaces (IIS/EIAJ) and over digital audio interfaces (IEC958).

Chapter 18, “I2C Modules”: This chapter provides the system configuration and protocol of the I^2C module, the memory map and register definitions, and a programming example.

Chapter 19, “Boot ROM”: This chapter describes the BootROM operation, the boot modes, and creation of record files.

Chapter 20, “Background Debug Mode (BDM) Interface”: This chapter details the MCF5251 hardware debug support. The topics discussed are real-time trace support, background debug mode (BDM), and real-time debug support. The memory map, register definitions, and Debug support operation are provided.

Chapter 21, “IEEE 1149.1 Test Access Port (JTAG)”: This chapter discussed the JTAG signal descriptions, TAP controller, memory map, register definitions, and how to disable the standard operation.

Chapter 22, “USB, ATA DMA, and Clock Integration Module”:This chapter includes the memory map, register definitions, and functional description of the integration module.

Chapter 23, “Advanced Technology Attachment Controller (ATA)”:This chapter discusses the modes of operation, signal descriptions, memory map, register definitions, timing parameters and functional description of the ATA interface.

Chapter 24, “Universal Serial Bus Interface”: This chapter describes the universal serial bus (USB) interface of the MCF5251. The content includes the operation, signal descriptions, host data structures, and host operations. Also provided is the device operational model and deviations from the host mode of operation.

Chapter 25, “FlexCAN Module”:This chapter discusses the modes of operation, signals, memory map, register definitions, and the functional and initialization sequence of the FlexCAN controller.

Chapter 26, “Real-Time Clock”: This chapter provides the external signal descriptions, memory map, register definitions, and functional descriptions of the Real-Time Clock.

Revision History

Table 1 summarizes revisions to this document since the previous release (Rev. 0).

Table 1. Revision History

Rev Number	Date of Release	Substantive Changes
Rev 1	1/2008	• Added USBMODE[ES] bit. • Added ADC max sampling frequency to section 12.4.1 • Clarified ADValue[OF] bit description. • 12.4.1 Added to paragraph "For a correct measurement..." that this applies only for the first conversion or when switching channels. • Reserved CSCRx[PS]=01 in table 10-5, since 8-bit port size is not possible on this device. • Added WIDESHIFT bit to FLASHMEDIACMD1 at location 22 for SD mode. Not present in MemoryStick mode. • Added missing overbar to $\overline{\text{HI\_Z}}$ signal throughout. • Fixed errors in 3 UART register diagrams. UMR2n register addresses are the same as UMR1n. UIVR2 was wrongly named UVR2 UOP12 was wrongly named UOP22 • Added buad rate calculation example, section 15.3.1.1.

Suggested Reading

Provide the full title, name of author, edition, and year of publication for any book you suggest as a supplement to this one.

PowerPC AIX Version 4 Application Binary Interface, 1st ed., April 1992.

The KornShell Command and Programming Language, Morris Bolsky and David Korn (Prentice Hall: 1989).

Conventions

This document uses the following notational conventions:

- `Courier monospaced type` indicate commands, command parameters, code examples, expressions, data types, and directives.
- *Italic type* indicates replaceable command parameters.
- All source code examples are in C.

Definitions, Acronyms, and Abbreviations

The following list defines the acronyms and abbreviations used in this document.

DSP	digital signal processor
JTAG	joint test access group
OnCE™	On-Chip Emulation
MIPS	million instructions per second
SRAM	static RAM
SDRAM block	DRAM memory selected by $\overline{\text{SD\_CS0}}$/GPIO60 signals. The base address of the block is programmed in the DRAM address and control register (DACR0).
SDRAM	RAMs that operate like asynchronous DRAMs but with a synchronous clock, a pipelined, multiple-bank architecture, and faster speed.
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 SDRAM component's bank select lines.

References

The following sources were referenced to produce this book:

1. Low-Level Software Design Document for the PowerPC Architectural Simulator (delivery date to be determined)
2. Requirements Chapter of the Software Project Management Plan for the PowerPC Microarchitectural Timing Simulator (delivery date to be determined)
3. PowerPC User Instruction Set Architecture, Book I, Version 1.00, 5/19/92 (subtitled "Work in Progress")
4. PowerPC Virtual Environment Architecture, Book II, Version 1.00, 5/19/92 (subtitled "Work in Progress")

Register Summary

Figure 1 shows the key to the register fields and Table 2 shows the register figure conventions.

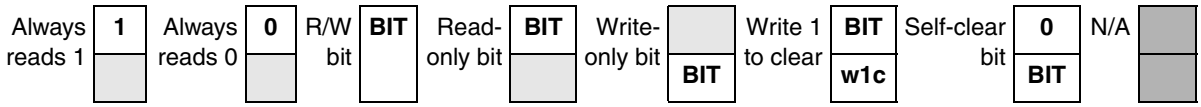


Figure 2. Key to Register Fields

Table 3. Register Figure Conventions

Convention	Description
	Depending on its placement in the read or write row, indicates that the bit is not readable or not writable.
FIELDNAME	Identifies the field. Its presence in the read or write row indicates that it can be read or written.
Register Field Types	
r	Read only. Writing this bit has no effect.
w	Write only.
rw	Standard read/write bit. Only software can change the bit's value (other than a hardware reset).
rwm	A read/write bit modified by a hardware in some fashion other than by a reset.
w1c	Write one to clear. A status bit that can be read, and is cleared by writing a one.
slfclr	Self-clearing bit. Writing a one has some effect on the module, but it always reads as zero.
Reset Values	
0	Resets to zero.
1	Resets to one.
—	Undefined at reset.
u	Unaffected by reset.
[*signal_name*]	Reset value is determined by polarity of indicated signal.

Chapter 1
MCF5251 Introduction

1.1 MCF5251 Overview

This chapter provides an overview of the MCF5251 ColdFire® processor and general descriptions of the MCF5251 features and modules.

The MCF5251 was designed as a system controller/decoder for compressed audio music players addressing both portable and automotive solutions supporting CD, HDD and USB based systems. The 32-bit ColdFire core with enhanced multiply and accumulate (eMAC) unit provides optimum performance and code density for the combination of control code and signal processing required for compressed audio decode, file management, and system control.

The MCF5251 is also an excellent general purpose system controller with over 125 Dhrystone 2.1 MIPS @ 140 MHz performance at a very competitive price. The integrated peripherals and eMAC allow the MCF5251 to replace both the microcontroller and the DSP in certain applications. Most peripheral pins can also be remapped as general purpose I/O pins.

Low power features include flexible PLL (with power-down mode) with dynamic clock switching, a hardwired CD ROM decoder, advanced 0.13um CMOS process technology, 1.2 V core power supply, and on-chip 128 Kbyte SRAM.

MP3 decode requires less than 20 MHz CPU bandwidth and runs in on-chip SRAM.

For additional information regarding software drivers and applications, refer to the MCF5251 website, http://www.freescale.com/coldfire.

1.2 MCF5251 Feature Introduction

The MCF5251 integrated microprocessor combines a Version 2 ColdFire processor core operating at 140 MHz with the following modules.

- USB 2.0 high-speed on-the-go (OTG) with integrated PHY
- Dedicated ATA hard disc interface
- Dedicated USB and ATA 16k SRAM with DMA support
- SmartMedia interface (including IDE and compact flash)
- Dual I^2C[1] controller
- Three UARTs
- NOR flash interface
- Supports 16-wide SDRAM memories

1. I^2C is a Philips proprietary bus

- Flash Memory Card Interface
- Serial Audio Interface which supports IIS and EIAJ audio protocols
- Digital audio transmitter (SPDIF) and two receivers compliant with IEC958 audio protocol
- Queued serial peripheral interface (QSPI) (master only)
- CD-ROM and CD-ROM XA block decoding and encoding function
- Two controller area network modules (FlexCAN)
- Embedded BDM debug port
- Integrated enhanced multiply and accumulate unit (eMAC)
- Unified cache system, 32-bits wide (8 Kbyte cache)
- 128 Kbytes zero-wait states SRAM, usable for code and data
- Interrupt controller with programmable interrupt priority
- DMA controller with 4 DMA channels
- On-chip real-time clock works with 32.768 kHz X-tal. Real-time clock has tamper detection functionality.
- Operates from crystal oscillator or external clock source
- Two timers
- 6-channel Analog/Digital Converter
- General Purpose I/O pins shared with other functions
- 1.2 V core, 3.3 V I/O
- Internal 1.2 V Linear regulator to power the core (configuration is optional)
- 225 pin MAPBGA package

1.3 MCF5251 Block Diagram

Figure 1-1 provides the block diagram of the MCF5251 device.

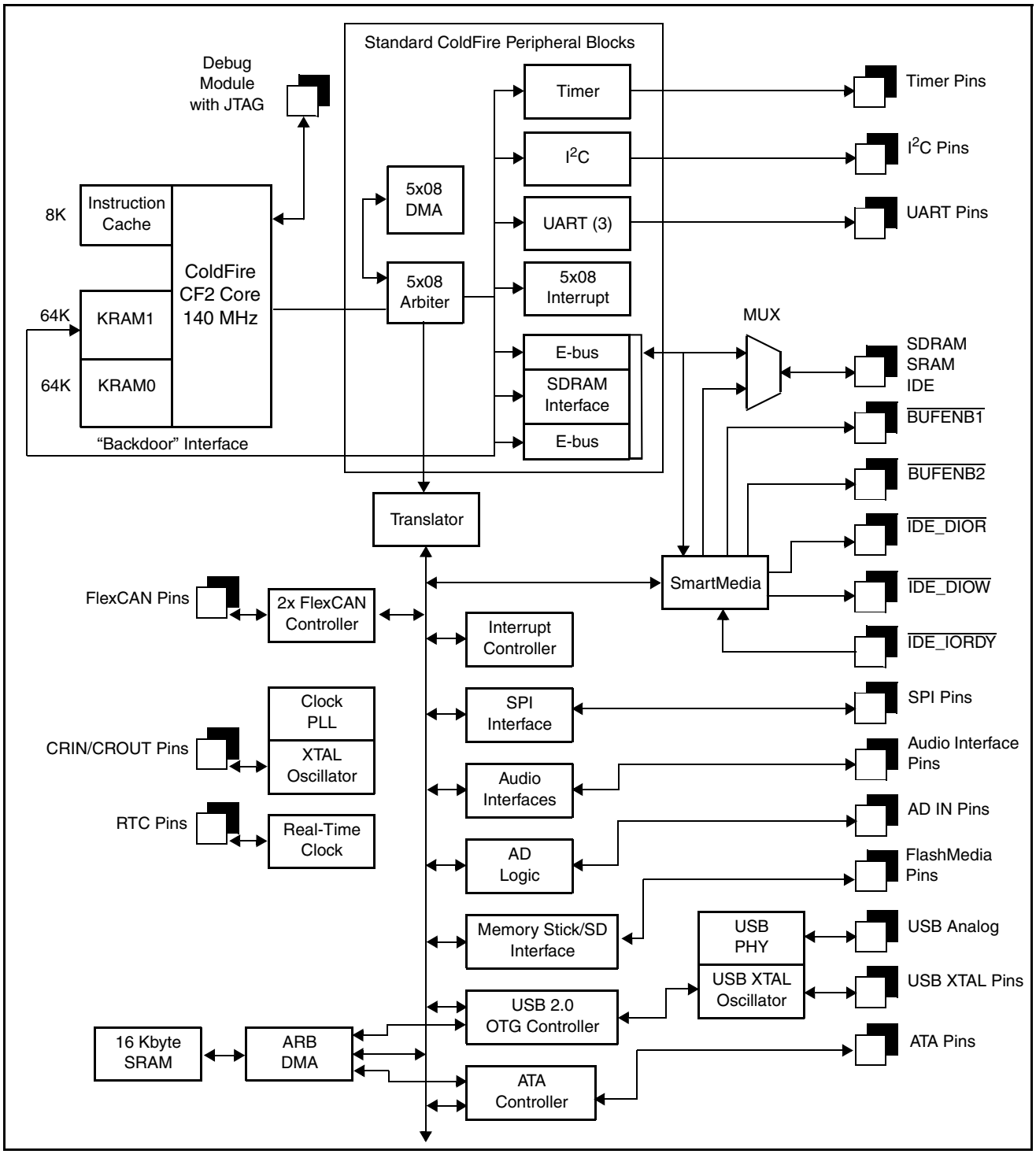


Figure 1-1. MCF5251 Block Diagram

1.4 MCF5251 Feature Details

The primary features of the MCF5251 integrated processor include the following:

- ColdFire CF2 Processor Core operating at 140 MHz
  - Clock-doubled Version 2 microprocessor core
  - 32-bit internal data bus, 16 bit external data bus
  - 16 user-visible, 32-bit general-purpose registers
  - Supervisor/user modes for system protection
  - Vector base register to relocate exception-vector table
  - Optimized for high-level language constructs
- DMA controller
  - Four fully programmable channels: Two dedicated to the audio interface module and two dedicated to the UART module. Any of the four channels can be associated to the ATA interface (External requests are not supported.)
  - Supports dual- and single-address transfers with 32-bit data capability
  - Two address pointers that can increment or remain constant
  - 16-/24-bit transfer counter
  - Operand packing and unpacking support
  - Auto-alignment transfers supported for efficient block movement
  - Supports bursting and cycle stealing
  - All channels support memory to memory transfers
  - Interrupt capability
  - Provides two clock cycle internal access
- Enhanced Multiply-accumulate Unit
  - Single-cycle multiply-accumulate operations for 32 x 32 bit and 16 x 16 bit operands
  - Support for signed, unsigned, integer, and fixed-point fractional input operands
  - Four 48-bit accumulators to allow the use of a 40-bit product
  - The addition of 8 extension bits to increase the dynamic number range
  - Fast signed and unsigned integer multiplies
- 8-Kbyte Direct Mapped Instruction Cache
  - Clocked at core clock frequency
  - Flush capability
  - Non-blocking cache provides fast access to critical code and data
- 128-Kbyte SRAM
  - Provides one-cycle access to critical code and data
  - Split into two banks, SRAM0 (64K), and SRAM1 (64K)
  - DMA requests to/from internal SRAM1 supported
- Crystal Trim

  - The XTRIM output can be used to trim an external crystal oscillator circuit which would allow lock with an incoming IEC958 or serial audio signal
- USB 2.0 high-speed on-the-go (OTG)
  - Compliant with OTG supplement to the USB 2.0 specification
  - Operates as high speed, full speed and low speed host, and as high speed and full speed device
  - Host negotiation protocol and session request protocol are implemented with software support, but also controllable by software.
  - On-chip USB 2.0 High-speed compatible PHY
- ATA Controller
  - Main use of this block is to interface with IDE hard disc drives and ATAPI optical disc drives.
  - Supports ATA6 pio modes 0, 1, 2, 3, and 4; multiword DMA modes 0, 1, and 2; ultra DMA modes 0, 1, 2, and 3.
- Twin Controller Area Network (CAN) 2.0B Communication Unit
  - The controller is a full implementation of the Bosch CAN protocol specification 2.0B, which supports both standard and extended message frames.
- Real-time Clock
  - Works with 32.768 kHz X-tal
  - Anti-tamper feature detects if clock was stopped by removing battery
- Audio Interfaces
  - SPDIF (IEC958) inputs and output
  - Three serial Philips IIS/Sony EIAJ interfaces
    - One with input and output, one with output only and one with input only (Two inputs, two outputs)
    - Master and Slave operation
- CD Text Interface
  - Allows the interface of CD subcode (transmitter only)
- Three Universal Asynchronous Receivers/Transmitters (UART*n*)
  - Full duplex operation
  - Baud-rate generator
  - Modem control signals: clear-to-send (CTS) and request-to-send (RTS) for UART0/1 only.
  - DMA interrupt capability
  - Processor-interrupt capability
- Queued Serial Peripheral Interface (QSPI)
  - Programmable queue to support up to 16 transfers without user intervention
  - Supports transfer sizes of 8 to 16 bits in 1-bit increments
  - Four peripheral chip-select lines for control of up to 15 devices
  - Supports Baud rates up to 17.5 Mbps at 140 MHz
  - Programmable delays before and after transfers

  - Programmable clock phase and polarity
  - Supports wraparound mode for continuous transfers
  - Master mode only
- Dual 16-bit General-purpose Multimode Timers
  - Clock source selectable from external, CPU clock/2 and CPU clock/32.
  - 8-bit programmable prescaler
  - 1 output
  - Processor-interrupt capability
  - 14.3 nS resolution with CPU clock at 140 MHz
- SmartMedia Interface
- Analog/Digital Converter
  - 12-Bit Resolution
  - 6 Muxed inputs
- Flash Memory Card Interface
  - Allows connection to Sony MemoryStick compatible devices
  - Support Secure Digital (SD) cards and other types of flash media (Multi-Media Card).
- Dual I^2C Interfaces
  - Interchip communication bus interface for EEPROMs, LCD controllers, A/D converters, keypads, CD-DSP's
  - Master and slave modes, support for multiple masters
  - Automatic interrupt generation with programmable level
- System debug support
  - Real-time instruction trace for determining dynamic execution path
  - Background debug mode (BDM) for debug features while halted
  - Debug exception processing capability
  - Real-time debug support
- System Interface
  - Glueless bus interface and DRAMC support for interfacing to 16-bit DRAM, SRAM, ROM, FLASH, and I/O devices
  - Three programmable chip-select signals for static memories or peripherals with programmable wait states and port sizes.
  - One dedicated chip select for 16-bit wide DRAM/SDRAM.
  - The device can boot from external memory or from its own internal boot ROM. If selected to boot from external memory (Flash / ROM) then CS0 is active after reset.
  - Programmable interrupt controller (low interrupt latency, seven external interrupt requests, programmable autovector generator)
  - Up to 57 programmable general-purpose outputs
  - Up to 60 programmable general-purpose inputs

  - IEEE 1149.1A Test (JTAG) Module
- Clocking
  - Clock-multiplied PLL, programmable frequency
- 1.2 V Core, 3.3 V I/O
- 225 pin BGA package (140 MHz)

1.5 MCF5251 Functional Overview

1.5.1 ColdFire CF2 Core

The ColdFire processor Version 2 (CF2) core consists of two independent, decoupled pipeline structures to maximize performance while minimizing core size.The instruction fetch pipeline (IFP) is a two-stage pipeline for prefetching instructions. The prefetched instruction stream is then gated into the two-stage operand execution pipeline (OEP), which decodes the instruction, fetches the required operands, and then executes the required function. Because the IFP and OEP pipelines are decoupled by an instruction buffer that serves as a FIFO queue, the IFP can prefetch instructions in advance of their actual use by the OEP, which minimizes time stalled waiting for instructions. The OEP is implemented in a two-stage pipeline featuring a traditional RISC data path with a dual-read-ported register feeding an arithmetic/logic unit (ALU).

1.5.2 DMA Controller

The MCF5251 provides four fully programmable DMA channels for quick data transfer. Single and dual address mode is supported with the ability to program bursting and cycle stealing. Data transfer is selectable as 8-, 16-, 32-, or 128-bits. Packing and unpacking is supported.

Two internal audio channels and two UART's can be used with the DMA channels. Any DMA channel can be used with the ATA interface. All channels can perform memory to memory transfers. The DMA controller has a user-selectable, 24- or 16-bit counter and a programmable DMA exception handler.

External requests are not supported.

1.5.3 Enhanced Multiply and Accumulate Module (eMAC)

The integrated eMAC unit provides a common set of DSP operations and enhances the integer multiply instructions in the ColdFire architecture. The eMAC provides functionality in three related areas:

1. Faster signed and unsigned integer multiplies
2. Multiply-accumulate operations supporting signed and unsigned operands
3. Miscellaneous register operations

Multiplies of 16x16 and 32x32 with 48-bit accumulates are supported in addition to a full set of extensions for signed and unsigned integers plus signed, fixed-point fractional input operands. The eMAC has a single-clock issue for 32x32-bit multiplication instructions and implements a four-stage execution pipeline.

1.5.4 Instruction Cache

The instruction cache improves system performance by providing cached instructions to the execution unit in a single clock cycle. The MCF5251 processor uses an 8K-byte, direct-mapped instruction cache to achieve 125 MIPS at 140 MHz. The cache is accessed by physical addresses, where each 16-byte line consists of an address tag and a valid bit. The instruction cache also includes a bursting interface for 16-bit and 8-bit port sizes to quickly fill cache lines.

1.5.5 Internal 128-Kbyte SRAM

The 128-Kbyte on-chip SRAM is split over two banks, SRAM0 (64K) and SRAM1 (64K). It provides single clock-cycle access for the ColdFire core. This SRAM can store processor stack and critical code or data segments to maximize performance. Memory in the second bank (SRAM1) can be accessed under DMA.

1.5.6 DRAM Controller

The MCF5251 DRAM controller provides a glueless interface for one bank of DRAM, and can address up to 32MB. The controller supports a 16-bit data bus. The controller operates in page mode, non-page mode, and burst-page mode and supports SDRAMs.

1.5.7 System Interface

The MCF5251 provides a glueless interface to 16-bit port size SRAM, ROM, and peripheral devices with independent programmable control of the assertion and negation of chip-select and write-enable signals.

The MCF5251 also supports bursting ROMs.

1.5.8 External Bus Interface

The bus interface controller transfers data between the ColdFire core or DMA and memory, peripherals, or other devices on the external bus. The external bus interface provides 23 address lines, a 16-bit data bus, Output Enable, and Read/Write signals. This interface implements an extended synchronous protocol that supports bursting operations.

1.5.9 USB 2.0 High-Speed On-The-Go

The USB module in the MCF5251 is used for communication to a PC or communication to slave devices, e.g. to download data from a hard disc player to a flash player, to a photo printer and so on. The USB supports full Host mode functionality. The USB supports the OTG supplement to the USB 2.0 specification. It operates as high speed, full speed and low speed host, and as high speed and full speed device. Host negotiation protocol (HNP) and session request protocol (SRP) are implemented with software support.

A USB 2.0 high-speed compatible PHY is integrated on-chip.

1.5.10 ATA Controller

The ATA block is an AT attachment host interface. Its main use is to interface with IDE hard disc drives and ATAPI optical disc drives. It interfaces with the ATA device over a number of ATA signals. The ATA interface is compliant to the ATA-6 standard, and supports PIO mode 0 to 4, multiword DMA mode 0, 1 and 2, and ultra DMA mode 0 to 3.

1.5.11 Two Controller Area Network (CAN) 2.0B Communication Unit

The two FlexCan modules are full implementation of the Bosch CAN protocol specification 2.0B, which supports both standard and extended message frames. The 32 message buffers that are supported, are stored in an embedded RAM.

1.5.12 Real-Time Clock

There is an on-chip real-time clock. The real-time clock needs an external 32.768 kHz crystal, and an external keep-alive battery. The real-time clock has an anti-tamper feature that will detect when the battery is removed.

1.5.13 Serial Audio Interfaces

The MCF5251 digital audio interface provides three serial Philips IIS/Sony EIAJ interfaces. One interface is a 4-pin (1 bit clock, 1 word clock, 1 data in, 1 data out), the other two interfaces are 3-pin (1 bit clock, 1 word clock, 1 data in or 1 data out). The serial interfaces have no limit on minimum sampling frequency. Maximum sampling frequency is determined by the maximum frequency on the bit clock input. (1/3 the frequency of the internal system clock.)

1.5.14 IEC958 Digital Audio Interfaces

The MCF5251 has two digital audio input interfaces, and one digital audio output interface. There are four digital audio input pins and two digital audio output pins. An internal multiplexer selects one of the four inputs to one of the two digital audio inputs.

One digital audio output carries the consumer “c” channel, the other carries the professional “c” channel.

The IEC958 outputs can take the output from the internal IEC958 generator, or multiplex out one of the four IEC958 inputs.

1.5.15 Audio Bus

The audio interfaces connect to an internal bus that carries all audio data. Each receiver places its received data on the audio bus and each transmitter takes data from the audio bus for transmission. Each transmitter has a source select register.

In addition to the audio interfaces, there are six CPU accessible registers connected to the audio bus. Three of these registers allow data reads from the audio bus and allow selection of the audio source. The other

three registers provide a write path to the audio bus and can be selected by transmitters as the audio source. Through these registers, the CPU has access to the audio samples for processing.

Audio can be routed from a receiver to a transmitter without the data being processed by the core so the audio bus can be used as a digital audio data switch. The audio bus can also be used for audio format conversion.

1.5.16 CD-ROM Encoder/Decoder

The MCF5251 is capable of processing CD-ROM sectors in hardware. Processing is compliant with CD-ROM and CD-ROM XA standards.

The CD-ROM decoder performs the following functions in hardware:

- Sector sync recognition
- Descrambling of sectors
- Verification of the CRC checksum for Mode 1, Mode 2 Form 1, and Mode 2 Form 2 sectors
- Third-layer error correction (ECC) is not performed in hardware. It is not essential to do Third-layer error correction but should it be determined for a particular application then a s/w Third-layer error correction is available.

The CD-ROM encoder performs the following functions in hardware:

- Sector sync recognition
- Scrambling of sectors
- Insertion of the CRC checksum for Mode 1, Mode 2 Form 1, and Mode 2 Form 2 sectors.
- Third-layer error encoding needs to be done in software. This can use approximately 5-10 MHz of performance for single-speed.

1.5.17 Three UART Modules

Three full-duplex UARTs with independent receive and transmit buffers are in this module. Data formats can be 5, 6, 7, or 8 bits with even, odd, or no parity, and up to 2 stop bits in 1/16 increments. Four-byte receive buffers and two-byte transmit buffers minimize CPU service calls. The Triple UART module also provides several error-detection and maskable-interrupt capabilities. Modem support includes request-to-send ($\overline{\text{RTS}}$) and clear-to-send ($\overline{\text{CTS}}$) lines for UART0/1. The third UART lacks request-to-send and clear-to-send lines.

The system clock provides the clocking function from a programmable prescaler. Users can select full duplex, auto-echo loopback, local loopback, and remote loopback modes. The programmable triple UARTs can interrupt the CPU on numerous events.

1.5.18 Queued Serial Peripheral Interface QSPI

The QSPI module provides a serial peripheral interface with queued transfer capability. It supports up to 16 stacked transfers at a time, making CPU intervention between transfers unnecessary. Transfers of up to

17.5 Mbits/second are possible at a CPU clock of 140 MHz. The QSPI supports master mode operation only.

1.5.19 Timer Module

The MCF5251 incorporates two independent, general-purpose 16-bit timers. The output of an 8-bit prescaler clocks each 16-bit timer. The prescaler input can be the system clock or the system clock divided by 16. Timer0 output pin is multiplexed with SDATAO1/TOUT0/GPIO18. Upon reset, this pin is programmed as SDATAO1. To use the TOUT0 pin function it is necessary to program the Pin Configuration register appropriately.

1.5.20 IDE Interface

The MCF5251 system bus allows connection of an IDE hard disk drive with a minimum of external hardware. The external hardware consists of bus buffers for address and data and are intended to reduce the load on the bus and prevent SDRAM and Flash accesses from propagating to the IDE bus. The control signals for the buffers are generated in the MCF5251.

1.5.21 Analog/Digital Converter (ADC)

The six channel ADC is based on the Sigma-Delta concept with 12-bit resolution. Both the analog comparator and digital sections are integrated in the MCF5251. An external integrator circuit (resistor/capacitor) is required which is driven by the ADC output. A interrupt is provided when the ADC measurement cycle is complete.

1.5.22 Flash Memory Card Interface

The interface is Sony® Memory Stick®, SecureDigital and Multi-Media card compatible. However, there is no hardware support for Sony MagicGate™.

1.5.23 I^2C Module

The two-wire I^2C bus interface, which is compliant with the Philips I^2C bus standard, is a bidirectional serial bus that exchanges data between devices. The I^2C bus minimizes the interconnection between devices in the end system and is best suited for applications that need occasional bursts of rapid communication over short distances among several devices. Bus capacitance and the number of unique addresses limit the maximum communication length and the number of devices that can be connected.

1.5.24 Chip-Selects

There are three programmable chip selects on the MCF5251:

- Three programmable chip-select outputs (CS0/CS4, CS1 and CS2) provide signals that enable glueless connection to external memory and peripheral circuits. The base address, access permissions, and automatic wait-state insertion are programmable with configuration registers. These signals also interface to 16-bit ports.

CS0 is intended to be used with an external boot ROM / Flash memory.
The MCF5251 can boot from its internal boot ROM, here CS0 is used internally, CS0/CS4 pin is then is configured as CS4. CS0 and CS4 cannot be used simultaneously.

- One dedicated chip select (CS2) is used for the IDE interface

1.5.25 GPIO Interface

Up to 60 General Purpose Inputs and up to 57 General Purpose Outputs are available. These are multiplexed with various other signals. Seven of the GPIO inputs have edge sensitive interrupt capability.

1.5.26 Interrupt Controller

The MCF5251 has a primary and a secondary interrupt controller. These interrupt controllers handle interrupts from all internal interrupt sources. In addition, there are 7 GPIOs where external interrupts can be generated on the rising or falling edge of the pin. All interrupts are autovectored and interrupt levels are programmable.

1.5.27 JTAG

To help with system diagnostics and manufacturing testing, the MCF5251 includes dedicated user-accessible test logic that complies with the IEEE 1149.1A standard for boundary scan testability, often referred to as Joint Test Action Group, or JTAG. For more information, refer to the IEEE 1149.1A standard. Freescale provides BSDL files for JTAG testing.

1.5.28 System Debug Interface

The ColdFire processor core debug interface supports real-time instruction trace and debug, plus background-debug mode. A background-debug mode (BDM) interface provides system debug.

In real-time instruction trace, four status lines provide information on processor activity in real time (PST pins). A four-bit wide debug data bus (DDATA) displays operand data and change-of-flow addresses, which helps track the CPU's dynamic execution path.

1.5.29 System Oscillator and PLL

The oscillator will operate from an external crystal connected across CRIN and CROUT. The circuit can also operate from an external clock connected to CRIN.

Typically, an external 16.92 MHz or 33.86 MHz clock input is used for CD R/W applications, while an 11.2896 MHz clock is more practical for Portable CD player applications. However, the on-chip programmable PLL, which generates the processor clock, allows the use of almost any low frequency external clock (5-35 MHz).

Two clock outputs (MCLK1 and MCLK2) are provided for use as Audio Master Clock. The output frequencies of both outputs are programmable to Fxtal, Fxtal/2, Fxtal/3, and Fxtal/4. The Fxtal/3 option is intended for use when the 33.86 MHz crystal option is used.

The MCF5251 supports voltage controlled crystal oscillator (VCXO) operation of the oscillator by means of a 16-bit pulse density modulation output. Using this mode, it is possible to lock the oscillator to the frequency of an incoming IEC958 or IIS signal. The maximum trim depends on the type and design of the oscillator. Typically a trim of +/- 100 ppm can be achieved with a crystal oscillator and over +/- 1000 ppm with an LC oscillator.

1.5.30 Sleep and Wake-Up Modes

The MCF5251 has a low power Sleep mode where all clocks are disabled and SRAM contents are maintained. Sleep mode is exited by taking the external Wake-up pin low. Sleep mode is selected by software control of a bit in the PLL register. When Sleep mode is exited code execution resumes from the next instruction.

1.5.31 Bootloader

The MCF5251 incorporates a ROM Bootloader, which enables booting from UART, I^2C, SPI or IDE devices.

1.5.32 Internal Voltage Regulator

An internal 1.2 V regulator can be used to supply the CPU and PLL sections of the MCF5251, reducing the number of external components required and allowing operation from a single supply rail, typically 3.3 volts. However, it must be noted that the internal regulator has an efficiency of less than 50%, and it is not intended for use in battery powered applications, where the use of a highly efficient external DC-DC converter would be more appropriate.

Chapter 2
Signal Description

2.1 Overview

This chapter describes the MCF5251 input and output signals. The signal descriptions as shown in Table 2-1 are grouped according to relevant functionality.

Table 2-1. MCF5251 Signal Index

Signal Name	Mnemonic	Function	Input/ Output	Reset State
Address	A[24:1] A[23]/GPO54	24 address lines—address 23 is multiplexed with GPO54 and address 24 is multiplexed with A20 (SDRAM access only).	Out	X
Read-write control	RW	Bus write enable—indicates if read or write cycle in progress.	Out	H
Output enable	$\overline{OE}$	Output enable for asynchronous memories connected to chip selects	Out	Negated
Data	D[31:16]	Data bus used to transfer word data	In/Out	Hi_Z
Synchronous row address strobe	$\overline{SDRAS}$/GPIO59	Row address strobe for external SDRAM	Out	Negated
Synchronous column address strobe	$\overline{SDCAS}$/GPIO39	Column address strobe for external SDRAM	Out	Negated
SDRAM write enable	$\overline{SDWE}$/GPIO38	Write enable for external SDRAM	Out	Negated
SDRAM upper byte enable	$\overline{SDUDQM}$/GPO53	Upper byte enable—indicates during write cycle if high byte is written.	Out	–
SDRAM lower byte enable	$\overline{SDLDQM}$/GPO52	Lower byte enable—indicates during write cycle if low byte is written.	Out	–
SDRAM chip selects	$\overline{SD\_CS0}$/GPIO60	SDRAM chip select	In/Out	Negated
SDRAM clock enable	BCLKE/GPIO63	SDRAM clock enable	Out	–
System clock	BCLK/GPIO40	SDRAM clock output	In/Out	–
ISA bus read strobe	$\overline{IDE\_DIOR}$/GPIO31 ($\overline{CS2}$)	1 ISA bus read strobe and 1 ISA bus write strobe—allow connection of an independent ISA bus peripheral, such as an IDE slave device.	In/Out	–
ISA bus write strobe	$\overline{IDE\_DIOW}$/GPIO32 ($\overline{CS2}$)		In/Out	–
ISA bus wait signal	$\overline{IDE\_IORDY}$/GPIO33	ISA bus wait line available for both busses	In/Out	–

Table 2-1. MCF5251 Signal Index (continued)

Signal Name	Mnemonic	Function	Input/ Output	Reset State
Chip Selects[2:0]	$\overline{CS0}$/$\overline{CS4}$ $\overline{CS1}$/QSPICS3/GPIO28	Chip selects bits 2 through 0—enable peripherals at programmed addresses. $\overline{CS0}$ provides boot ROM selection.	Out In/Out	negated
Buffer enable 1	$\overline{BUFENB1}$/GPIO29	Two programmable buffer enables—allow seamless steering of external buffers to split data and address bus in sections.	In/Out	–
Buffer enable 2	$\overline{BUFENB2}$/GPIO30		In/Out	–
Transfer acknowledge	$\overline{TA}$/GPIO12	Transfer Acknowledge signal.	In/Out	–
Wake Up	$\overline{WAKEUP}$/GPIO21	Wake-up signal input	In	–
Serial Clock Line	SCL0/SDATA1_BS1/GPIO41 SCL1/TXD1/GPIO10	Clock signal for Dual I^2C module operation	In/Out	–
Serial Data Line	SDA0/SDATA3/GPIO42 SDA1/RXD1/GPIO44	Serial data port for second I^2C module operation	In/Out	–
Receive Data	SDA1/RXD1/GPIO44 RXD0/GPIO46 EF/RXD2/GPIO6	Receive serial data input for UART	In	–
Transmit Data	SCL1/TXD1/GPIO10 TXD0/GPIO45 XTRIM/TXD2/GPIO0	Transmit serial data output for UART	Out	–
Request-To-Send	DDATA3/$\overline{RTS0}$/GPIO4 DDATA1/$\overline{RTS1}$/SDATA2_BS2/GPIO2	Signals sent from UART0/1 that it is ready to receive data	Out	–
Clear-To-Send	DDATA2/$\overline{CTS0}$/GPIO3 DDATA0/$\overline{CTS1}$/SDATA0_SDIO1/GPIO1	Signals sent to UART0/1 that data can be transmitted to peripheral	In	–
Timer Output	SDATAO1/TOUT0/GPIO18	Capability of output waveform or pulse generation	Out	–
IEC958 inputs	EBUIN1/GPIO36 EBUIN2/SCLKOUT/GPIO13 EBUIN3/CMD_SDIO2/GPIO14 QSPICS0/EBUIN4/GPIO15	Audio interfaces to IEC958 inputs	In	–
IEC958 outputs	EBUOUT1/GPIO37 QSPICS1/EBUOUT2/GPIO16	Audio interfaces to IEC958 outputs	Out	–
Serial data in	SDATAI1/GPIO17 SDATAI3/GPIO8	Audio interfaces to serial data inputs	In	–
Serial data out	SDATAO1/TOUT0/GPIO18 SDATAO2/GPIO34	Audio interfaces to serial data outputs	In/Out Out	–
Word clock	LRCK1/GPIO19 LRCK2/GPIO23 LRCK3/AUDIOCLK/GPIO43	Audio interfaces to serial word clocks	In/Out	–
Bit clock	SCLK1/GPIO20 SCLK2/GPIO22 SCLK3/GPIO35	audio interfaces to serial bit clocks	In/Out	–

Table 2-1. MCF5251 Signal Index (continued)

Signal Name	Mnemonic	Function	Input/ Output	Reset State
Serial input	EF/RXD2/GPIO6	Error flag serial in	In/Out	–
Serial input	CFLG/GPIO5	C-flag serial in	In/Out	–
Subcode clock	RCK/QSPIDIN/QSPIDOUT/ GPIO26	Audio interfaces to subcode clock	In/Out	–
Subcode sync	QSPIDOUT/SFSY/GPIO27	Audio interfaces to subcode sync	In/Out	–
Subcode data	QSPICLK/SUBR/GPIO25	Audio interfaces to subcode data	In/Out	–
Clock frequency trim	XTRIM/TXD2/GPIO0	Clock trim control	Out	–
Audio clocks out	MCLK1/GPIO11 QSPICS2/MCLK2/GPIO24	DAC output clocks	Out	–
Audio clock in	LRCK3/AUDIOCLK/GPIO43	Optional audio clock input		–
MemoryStick/ SecureDigital interface	EBUIN3/CMD_SDIO2/GPIO14	Secure Digital command lane— MemoryStick interface 2 data I/O	In/Out	–
	EBUIN2/SCLKOUT/GPIO13	Clock out for both MemoryStick interfaces and for Secure Digital	In/Out	–
	DDATA0/$\overline{\text{CTS1}}$/SDATA0_SDIO1/GPIO1	SecureDigital serial data bit 0— MemoryStick interface 1 data I/O	In/Out	–
	SCL0/SDATA1_BS1/GPIO41	SecureDigital serial data bit 1— MemoryStick interface 1 strobe	In/Out	–
	DDATA1/$\overline{\text{RTS1}}$/SDATA2_BS2/GPIO2	SecureDigital serial data bit 2— MemoryStick interface 2 strobe Reset output signal	In/Out	–
	SDA0/SDATA3/GPIO42	SecureDigital serial data bit 3	In/Out	–
AT attachment interface (IDE interface)	ATA_DIOW	ATA write strobe signal	Out	–
	ATA_DIOR	ATA read strobe signal	Out	–
	ATA_IORDY	ATA I/O ready input	In	–
	ATA_DMARQ	ATA DMA request	In	–
	ATA_DMACK	ATA DMA acknowledge	Out	–
	ATA_INTRQ	ATA interrupt request	In	–
	ATA_CS0	ATA chip select 0	Out	–
	ATA_CS1	ATA chip select 1	Out	–
	ATA_A[2:0]	3-bit ATA address bus	Out	–
	ATA_D[15:0]	16-bit ATA data bus	In/Out	–

Table 2-1. MCF5251 Signal Index (continued)

Signal Name	Mnemonic	Function	Input/ Output	Reset State
CAN interface	CAN0_TX	CAN 0 transmit	Out	–
	CAN0_RX	CAN 0 receive	In	–
	CAN1_TX	CAN 1 transmit	Out	–
	CAN1_RX	CAN 1 receive	In	–
USB PHY interface	USBVBUS	USB Vbus input	In	–
	USBID	USB ID input	In	–
	USBRES	USB current programming resistor pin	Analog	–
	USBDN	USB DM signalling line	In/Out	–
	USBDP	USB DP signalling line	In/Out	–
USB oscillator	USB_CRIN USB_CROUT	Connections for USB oscillator crystal (24 MHz)	In Out	–
RTC oscillator	RTC_CRIN RTCCROUT	Connections for real-time clock crystal (32.768 kHz)	In Out	–
AD IN	ADIN0/GPI52 ADIN1/GPI53 ADIN2/GPI54 ADIN3/GPI55 ADIN4/GPI56 ADIN5/GPI57	Analog-to-Digital Converter input signals	In	–
AD OUT	ADREF ADOUT/SCLK4/GPIO58	Analog-to-Digital Converter output signal—connects to ADREF via integrator network.	In/Out	–
QSPI clock	QSPICLK/SUBR/GPIO25	QSPI clock signal	In/Out	–
QSPI data in	RCK/QSPIDIN/QSPIDOUT/GPIO26	QSPI data input	In/Out	–
QSPI data out	RCK/QSPIDIN/QSPIDOUT/GPIO26 QSPIDOUT/SFSY/GPIO27	QSPI data out	In/Out	–
QSPI chip selects	QSPICS0/EBUIN4/GPIO15 QSPICS1/EBUOUT2/GPIO16 QSPICS2/MCLK2/GPIO24 $\overline{CS1}$/QSPICS3/GPIO28	QSPI chip selects	In/Out	–
System oscillator in	CRIN	System input	In	–
System oscillator out	CROUT	System output	Out	–
Reset In	$\overline{RSTI}$	Processor reset input	In	–
Freescale Test Mode	TEST[2:0]	TEST pins.	In	–
Linear regulator output	LINOUT	Output of 1.2 V to supply core	Out	–
Linear regulator input	LININ	Input, typically I/O supply (3.3V)	In	–
Linear regulator ground	LINGND			–

Table 2-1. MCF5251 Signal Index (continued)

Signal Name	Mnemonic	Function	Input/ Output	Reset State
High Impedance	HI_Z	Assertion tri-states output signal pins	In	
Debug Data	DDATA0/CTS1/SDATA0_SDIO1/GPIO1 DDATA1/RTS1/SDATA2_BS2/GPIO2 DDATA2/CTS0/GPIO3 DDATA3/RTS0/GPIO4	Display of captured processor data and break-point statuses	In/Out	Hi_Z
Processor Status	PST0/GPIO50 PST1/GPIO49 PST2/INTMON2/GPIO48 PST3/INTMON1/GPIO47	Indication of internal processor status.	In/Out	Hi_Z
Processor clock	PSTCLK/GPIO51	Processor clock output	Out	–
Test Clock	TCK	Clock signal for IEEE 1149.1A JTAG	In	–
Test Reset/ Development Serial Clock	TRST/DSCLK	Multiplexed signal that is asynchronous reset for JTAG controller. Also, clock input for debug module.	In	–
Test Mode Select/Break Point	TMS/BKPT	Multiplexed signal that is test mode select in JTAG mode and a hardware break-point in debug mode	In	–
Test Data Input/ Development Serial Input	TDI/DSI	Multiplexed serial input for the JTAG or background debug module.	In	–
Test Data Output/Development Serial Output	TDO/DSO	Multiplexed serial output for the JTAG or background debug module	Out	–

2.2 GPIO

Many pins have an optional GPIO function.

- General purpose input is always active, regardless of state of pin.
- General purpose output or primary output is determined by the appropriate setting of the Pin Multiplex Control Registers, GPIO-FUNCTION, GPIO1-FUNCTION and PIN-CONFIG.
- At Power-on reset all pins are set to their primary function.

2.3 MCF5251 Bus Signals

The signals discussed in this section provide the external bus interface to the MCF5251.

2.3.1 Address Bus

The address bus provides the address of the byte or most significant byte of the word or longword being transferred.The address lines also serve as the DRAM address pins, providing multiplexed row and column address signals.

Bits 23 down to 1 and 24 of the address are available. A24 is intended to be used with 256 Mbit DRAM's.

Signals are named:

- A[23:1]
- A20/24

2.3.2 Read-Write Control

This signal indicates during any bus cycle whether a read or write is in progress. A low is write cycle and a high is a read cycle.

2.3.3 Output Enable

The $\overline{OE}$ signal is intended to be connected to the output enable of asynchronous memories connected to chip selects. During bus read cycles, the ColdFire processor will drive $\overline{OE}$ low.

2.3.4 Data Bus

The data bus (D[31:16]) is bi-directional and non-multiplexed. Data is registered by the MCF5251 on the rising clock edge. The data bus uses a default configuration if none of the chip-selects or DRAM bank match the address decode. All 16 bits of the data bus are driven during writes, regardless of port width or operand size.

2.3.5 Transfer Acknowledge

The $\overline{TA}$/GPIO12 pin is the transfer acknowledge signal.

2.4 SDRAM Controller Signals

The following SDRAM signals provide a glueless interface to external SDRAM. An SDRAM width of 16 bits is supported and can access as much as 32MBs of memory. ADRAMs are not supported.

Table 2-2. SDRAM Controller Signals

SDRAM Signal	Description
Synchronous DRAM row address strobe	The $\overline{SDRAS}$/GPIO59 active low pin provides a seamless interface to the RAS input on synchronous DRAM
Synchronous DRAM column address strobe	The $\overline{SDCAS}$/GPIO39 active low pin provides a seamless interface to CAS input on synchronous DRAM.
Synchronous DRAM write	The $\overline{SDWE}$/GPIO38 active-low pin is asserted to signify that a SDRAM write cycle is underway. This pin outputs logic '1' during read bus cycles.
Synchronous DRAM chip enable	The $\overline{SD\_CS0}$/GPIO60 active-low output signal is used during synchronous mode to route directly to the chip select of a SDRAM device.
Synchronous DRAM UDQM and LQDM signals	The DRAM byte enables UDMQ and LDQM are driven by the $\overline{SDUDQM}$/GPO53 and $\overline{SDLDQM}$/GPO52 byte enable outputs.

Table 2-2. SDRAM Controller Signals

SDRAM Signal	Description
Synchronous DRAM clock	The DRAM clock is driven by the BCLK/GPIO40 signal.
Synchronous DRAM clock enable	The BCLKE active high output signal is used during synchronous mode to route directly to the SCKE signal of external SDRAMs. This signal provides the clock enable to the SDRAM.

2.5 Chip Selects

There are three chip select outputs on the MCF5251 device. $\overline{\text{CS0}}$/$\overline{\text{CS4}}$ and $\overline{\text{CS1}}$/QSPI_CS3/GPIO28 and CS2 which is associated with the IDE interface read and write strobes—IDE_DIOR and IDE_DIOW.

CS0 and CS4 are multiplexed. The MCF5251 has the option to boot from an internal Boot Rom. The function of the CS0/CS4 pin is determined by the boot mode. When the device is booted from internal ROM, the internal ROM is accessed with CS0 (required for boot) and the CS0/CS4 pin is driven by CS4. When the device is booted from external ROM / Flash, the CS0/CS4 pin is driven by CS0 and the internal ROM is disabled.

The active low chip selects can be used to access asynchronous memories. The interface is glueless.

2.6 ISA Bus

The MCF5251 supports an ISA bus. Using the ISA bus protocol, reads and writes for one ISA bus peripheral is possible. $\overline{\text{IDE\_DIOR}}$/GPIO31 and $\overline{\text{IDE\_DIOW}}$/GPIO32 are the read and write strobe. The peripheral can insert wait states by pulling IDE_IORDY/GPIO33.

CS2 is associated with the IDE_DIOR and IDE_DIOW.

2.7 Bus Buffer Signals

As the MCF5251 has a quite complicated slave bus, with the possibility of having DRAM, asynchronous memories and an ISA peripherals on the bus, it may become necessary to introduce a buffer on the bus. The MCF5251 has a glueless interface to steer these bus buffers with 2 bus buffer output signals $\overline{\text{BUFENB1}}$/GPIO29 and $\overline{\text{BUFENB2}}$/GPIO30.

2.8 I^2C Module Signals

There are two I^2C interfaces on this device.

The I^2C module acts as a two-wire, bidirectional serial interface between the MCF5251 processor and peripherals with an I^2C interface (e.g., LED controller, A-to-D converter, D-to-A converter). When devices connected to the I^2C bus drive the bus, they will either drive logic-0 or high-impedance. This can be accomplished with an open-drain output.

Table 2-3. I²C Module Signals

I²c Module Signal	Description
I²C Serial Clock	The SCL0/SDATA1_BS1/GPIO41, and SCL1/TXD1/GPIO10 bidirectional signals are the clock signal for first and second I²C module operation. The I²C module controls this signal when the bus is in master mode; all I²C devices drive this signal to synchronize I²C timing. Signals are multiplexed.
I²C Serial Data	The SDA0/SDATA3/GPIO42 and SDA1/RXD1/GPIO44 bidirectional signals are the data input/output for the first and second serial I²C interface. Signals are multiplexed.

2.9 Serial Module Signals

The following signals transfer serial data between the three UART modules and external peripherals.

Table 2-4. Serial Module Signals

Serial Module Signal	Description
Receive Data	The RXD0/GPIO46, SDA1/RXD1/GPIO44, and EF/RXD2/GPIO6 are the inputs on which serial data is received by the UART. Data is sampled on RxD[2:0] on the rising edge of the serial clock source, with the least significant bit received first.
Transmit Data	The UART transmits serial data on the TXD0/GPIO45, SCL1/TXD1/GPIO10, and XTRIM/TXD2/GPIO0 output signals. Data is transmitted on the falling edge of the serial clock source, with the least significant bit transmitted (LSB) first. When no data is being transmitted or the transmitter is disabled, these two signals are held high. TxD[2:0] are also held high in local loopback mode.
Request To Send	The DDATA3/$\overline{\text{RTS0}}$/GPOI4 and DDATA1/$\overline{\text{RTS1}}$/SDATA2_BS2/GPIO2 request-to-send outputs indicate to the peripheral device that UART0/1 are ready to send data and requires a clear-to-send signal to initiate transfer. The third UART lacks flow control using RTS/CTS.
Clear To Send	Peripherals drive the DDATA2/$\overline{\text{CTS0}}$/GPIO3 and DDATA0/$\overline{\text{CTS1}}$/SDATA0_SDIO1/GPIO1 inputs to indicate to the MCF5251 serial module that it can begin data transmission. The third UART lacks flow control using RTS/CTS.

2.10 Timer Module Signals

The following signal provides an external interface to Timer0. One 16-bit timers can trigger external events or both 16-bit timers can trigger internal interrupts.

Table 2-5. Timer Module Signals

Timer Signal	Description
Timer Output	The SDATAO1/TOUT0/GPIO18 programmable output pulse or toggle on various timer events.

2.11 Serial Audio Interface Signals

The following signals provide the external audio interface.

Table 2-6. Serial Audio Interface Signals

Audio Interface Signal	Description
Serial audio bit clock	The SCLK1/GPIO20, SCLK2/GPIO22, and SCLK3/GPIO35 multiplexed pins can serve as general purpose I/Os or serial audio bit clocks. As bit clocks, these bidirectional pins can be programmed as outputs to drive their associated serial audio (IIS) bit clocks. Alternately, these pins can be programmed as inputs when the serial audio bit clocks are driven internally. The functionality is programmed within the Audio module. During reset, these pins are configured as input serial audio bit clocks.
Serial audio word clock	The LRCK1/GPIO19, LRCK2/GPIO23, and LRCK3/AUDIOCLK/GPIO43 multiplexed pins can serve as general purpose I/Os or serial audio word clocks. As word clocks, the bidirectional pins can be programmed as inputs to drive their associated serial audio word clock. Alternately, these pins can be programmed as outputs when the serial audio word clocks are derived internally. The functionality is programmed within the Audio module. During reset, these pins are configured as input serial audio word clocks. LRCK3/AUDIOCLK/GPIO43 can be used as the external audio clock input. If the core clock chosen to be non-audio specific.
Serial audio data in	The SDATAI1/GPIO17 and SDATAI3/GPIO8 multiplexed pins can serve as general purpose I/Os or serial audio inputs. As serial audio inputs the data is sent to interfaces 1and 3 respectively. During reset, the pins are configured as serial data inputs.
Serial audio data out	SDATAO1/TOUT0/GPIO18 AND SDATAO2/GPIO34 multiplexed pins can serve as general purpose I/Os or serial audio outputs. During reset, the pins are configured as serial data outputs.
Serial audio error flag	The EF/GPIO6 multiplexed pin can serve as general purpose I/Os or error flag input. As error flag input, this pin will input the error flag delivered by the CD-DSP. EF/GPIO6 is only relevant for serial interface SDATAI1.
Serial audio CFLG	The CFLG/GPIO5 multiplexed pin can serve as general purpose I/O or CFLG input. As CFLG input, the pin will input the CFLG flag delivered by the CD-DSP. CFLG/GPIO5 is only relevant for serial interface SDATAI1.

2.12 Digital Audio Interface Signals

Table 2-7. Digital Audio Interface Signals

Digital Audio Interface Signal	Description
Digital audio in	The EBUIN1/GPIO36, EBUIN2/SCLKOUT/GPIO13, EBUIN3/CMD_SDIO2/GPIO14, and QSPICS0/EBUIN4/GPIO15 multiplexed signals can serve as general purpose input or can be driven by various digital audio (IEC958) input sources. Both functionalities are always active. Input chosen for IEC958 receiver is programmed within the audio module. Input value on the 4 pins can always be read from the appropriate gpio register.
Digital audio out	The EBUOUT1/GPIO37 and QSPICS1/EBUOUT2/GPIO16 multiplexed pins can serve as general purpose I/O or as digital audio (IEC958) output. EBUOUT1 is digital audio out for consumer mode, EBUOUT2 is digital audio out for professional mode. During reset, the pin is configured as a digital audio output.

2.13 Subcode Interface

There is a 3-line subcode interface on the MCF5251. This 3-line subcode interface allows the device to format and transmit subcode in EIAJ format to a CD channel encoder device. The three signals are described in Table 2-8.

Table 2-8. Subcode Interface Signal

Subcode Interface Signal	Description
RCK/QSPIDIN/QSPIDOUT/GPIO26	Subcode clock input. When pin is used as subcode clock, this pin is driven by the CD channel encoder.
QSPIDOUT/SFSY/GPIO27	Subcode sync output. This signal is driven high if a subcode sync needs to be inserted in the EFM stream.
QSPICLK/SUBR/GPIO25	Subcode data output. This signal is a subcode data out pin.

2.14 Analog to Digital Converter (ADC)

The ADOUT signal on the ADOUT/SCLK4/GPIO58 pin provides the reference voltage in PWM format. Therefore this output requires an external integrator circuit (resistor/capacitor) to convert it to a DC level to be input to the ADREF pin.

The six AD inputs are each fed to their own comparator the reference input to each (ADREF) is then multiplexed as only one AD comparison can be made at any one time.

NOTE

To use the ADINx as General Purpose inputs (rather than the analog function) it is necessary to generate a fixed comparator voltage level of VDD/2. This is accomplished by a potential divider network connected to the ADREF pin. However in portable applications where stand-by power consumption is important, the current taken by the divider network (in stand-by mode) can be excessive. Therefore it is possible to generate a VDD/2 voltage by selecting SCLK4 output mode and feeding this clock signal (which is 50% duty cycle) through an external integration circuit. This generates a voltage level equal to VDD/2, however when stand-by mode is selected it is disabled.

2.15 Secure Digital / Memory Stick Card Interface

The device has a versatile flash card interface that supports both SecureDigital and MemoryStick cards. The interface can either support one SecureDigital or two MemoryStick cards. No mixing of card types is possible. Table 2-9 gives the pin descriptions.

Table 2-9. Flash Memory Card Signals

Flash Memory Signal	Description
EBUIN2/SCLKOUT/GPIO13	Clock out for both MemoryStick interfaces and for SecureDigital.
EBUIN3/CMD_SDIO2/GPIO14	Secure Digital command line. MemoryStick interface 2 data I/O.

Table 2-9. Flash Memory Card Signals (continued)

Flash Memory Signal	Description
DDATAO/$\overline{\text{CTS1}}$/SDATA0_SDIO1/GPIO1	Secure Digital serial data bit 0. MemoryStick interface 1 data I/O.
SCL0/SDATA1_BS1/GPIO41	Secure Digital serial data bit 1. MemoryStick interface 1 strobe.
DDATA1/$\overline{\text{RTS1}}$/SDATA2_BS2/GPIO2	Secure Digital serial data bit 2. MemoryStick interface 2 strobe.
SDA0/SDATA3/GPIO42	Secure Digital serial data bit 3.

2.16 Queued Serial Peripheral Interface (QSPI)

The QSPI interface is a high-speed serial interface allowing transmit and receive of serial data. Pin descriptions are given in Table 2-10.

Table 2-10. Queued Serial Peripheral Interface (QSPI) Signals

QSPI Signal	Description
QSPICLK/SUBR/GPIO25	Multiplexed signal IIC interface clock or QSPI clock output Function select is done via pin configuration register.
RCK/QSPIDIN/QSPIDOUT/GPIO26	Multiplexed signal IIC interface data or QSPI data input. Function select is done via pin configuration register.
RCK/QSPIDIN/QSPIDOUT/GPIO26 QSPIDOUT/SFSY/GPIO27	QSPI data input/output
QSPICLK/SUBR/GPIO25	QSPI clock signal
QSPICS0/EBUIN4GPIO15	4 different QSPI chip selects
QSPICS1/EBUOUT2/GPIO16	
QSPICS2/MCLK2/GPIO24	
CS1/QSPICS3/GPIO28	

2.17 ATA Interface

The ATA interface is a high-speed interface to IDE hard disc drives or to ATAPI optical disc drives. Besides this interface, the device also has a ISA bus interface. However, the ISA bus is intended for slow peripherals. It only supports PIO mode transfers. The AT attachment interface supports PIO transfers and the faster multiword DMA and ultra DMA transfers. Pin descriptions are given in Table 2-1.

2.18 Two Controller Area Network (CAN) Communication Modules

The two FlexCan modules are full implementation of the Bosch CAN protocol specification 2.0B. Pin descriptions are given in Table 2-1.

2.19 USB Controller

The MCF5251 is fitted with an on-chip USB controller.

2.19.1 USB PHY Interface Including Oscillator

There is an integrated on-chip USB PHY. Pins are described in Table 2-11.

Table 2-11. USB PHY Interface Pins

USB PHY Signal	Description
USBVBUS	Vbus input/output
USBRES	Current programming resistor pin of 6.05k Ω having a 1% tolerance
USBID	USB ID input
USBDN	USB dm signalling line
USBDP	USB dp signalling line
USB_CRIN	A 24 MHz X-tal needs to be connected between these 2 pins
USB_CROUT	

2.20 Real-Time Clock

There is a real-time clock integrated in the device. Pins are described in Table 2-12.

Table 2-12. Real-Time Clock (RTC) Pins

Real-Time Clock Signal	Description
RTC_CRIN	Connect a real-time clock crystal (32.768 kHz) between these 2 pins.
RTCCROUT	

2.21 Crystal Trim

The XTRIM/TXD2/GPIO0 output produces a pulse-density modulated phase/frequency difference signal to be used after low-pass filtering to control varicap-voltage to control crystal oscillation frequency. This will lock the crystal to the incoming digital audio signal.

2.22 Clock Out

The MCLK1/GPIO11 and QSPI_CS2/MCLK2/GPIO24 can serve as DAC clock outputs. When programmed as DAC clock outputs, these signals are directly derived from the crystal oscillator or clock Input (CRIN).

2.23 Debug and Test Signals

These signals interface with external I/O to provide processor debug and status signals.

2.23.1 Test Mode

The TEST[2:0] inputs are used for various manufacturing and debug tests. For normal mode TEST [2:1] should be ways be tied low. TEST0 should be set high for BDM debug mode and set low for JTAG mode.

2.23.2 High Impedance

The assertion of $\overline{\text{HI\_Z}}$ will force all output drivers to a high-impedance state. The timing on $\overline{\text{HI\_Z}}$ is independent of the clock.

NOTE

JTAG operation will override the $\overline{\text{HI\_Z}}$ pin.

2.23.3 Processor Clock Output

The internal PLL generates this PSTCLK/GPIO51 and output signal, and is the processor clock output that is used as the timing reference for the Debug bus timing (DDATA[3:0] and PST[3:0]). The PSTCLK/GPIO51 is at the same frequency as the core processor.

2.23.4 Debug Data

The debug data pins, DDATA0/CTS1/SDATA0_SDIO1/GPIO1, DDATA1/$\overline{\text{RTS1}}$/SDATA2_BS2/GPIO2, DDATA2/CTS0/GPIO3, and DDATA3/$\overline{\text{RTS0}}$/GPIO4, are four bits wide. This nibble-wide bus displays captured processor data and break-point status. Refer to Chapter 20, “Background Debug Mode (BDM) Interface,” for additional information on this bus.

2.23.5 Processor Status

The processor status pins, PST0/GPIO50, PST1/GPIO49, PST2/INTMON/GPIO48, and PST3/INTMON/GPIO47, indicate the MCF5251 processor status. During debug mode, the timing is synchronous with the processor clock (PSTCLK) and the status is not related to the current bus transfer. Table 2-13 shows the encodings of these signals.

Table 2-13. Processor Status Signal Encodings

PST[3:0]		Definition
(Hex)	(Binary)	
$0	0000	Continue execution
$1	0001	Begin execution of an instruction
$2	0010	Reserved
$3	0011	Entry into user-mode
$4	0100	Begin execution of PULSE and WDDATA instructions
$5	0101	Begin execution of taken branch or Synch_PC[1]
$6	0110	Reserved

Table 2-13. Processor Status Signal Encodings (continued)

PST[3:0]		Definition
(Hex)	(Binary)	
$7	0111	Begin execution of RTE instruction
$8	1000	Begin 1-byte data transfer on DDATA
$9	1001	Begin 2-byte data transfer on DDATA
$A	1010	Begin 3-byte data transfer on DDATA
$B	1011	Begin 4-byte data transfer on DDATA
$C	1100	Exception processing[2]
$D	1101	Emulator mode entry exception processing[2]
$E	1110	Processor is stopped, waiting for interrupt[2]
$F	1111	Processor is halted[2]

1 Rev. B enhancement.

2 These encodings are asserted for multiple cycles.

2.24 BDM/JTAG Signals

The MCF5251 complies with the IEEE 1149.1A JTAG testing standard. The JTAG test pins are multiplexed with background debug pins. See Chapter 20, "Background Debug Mode (BDM) Interface," for details.

2.25 Clock and Reset Signals

These signals configure the MCF5251 and provide interface signals to the external system.

2.25.1 Reset In

Asserting $\overline{\text{RSTI}}$ causes the MCF5251 to enter reset exception processing. When $\overline{\text{RSTI}}$ is recognized, the data bus is tri-stated.

2.25.2 System Bus Input

MCF5251 includes on-chip crystal oscillator. The crystal must be connected between CRIN and CROUT.

An externally generated clock signal can also be used and should be connected directly to the CRIN pin.

2.26 Wake-Up Signal

To exit power down mode, apply a LOW level to the $\overline{\text{WAKEUP}}$/GPIO21 input pin.

2.27 On-Chip Linear Regulator

The MCF5251 includes an on-chip linear regulator. This regulator provides an 1.2 V output which is intended to be used to power the MCF5251 core. Three pins are associated with this function; LININ, LINOUT, and LINGND. Typically, LININ would be fed by the I/O (PAD) supply (3.3 V) with separate filtering recommended to provide some isolation between the I/O and the core.

In portable solutions, this linear regulator may not be efficient enough. In this case, we would expect the 1.2 V supply to be generated externally, possibly by a highly efficient DC-DC convertor.

If not used, leave pins unconnected.

Chapter 3
ColdFire Core

This chapter provides an overview of the microprocessor core of the MCF5251. The chapter describes the CF2 memory map and register description as it is implemented on the MCF5251. It also includes a full description of exception handling, data formats, an instruction set summary, and a table of instruction timings. For detailed information on instructions, see the *ColdFire Family Programmer's Reference Manual*.

3.1 Processor Pipelines

Figure 3-1 shows a block diagram of the processor pipelines of a CF2 ColdFire core.

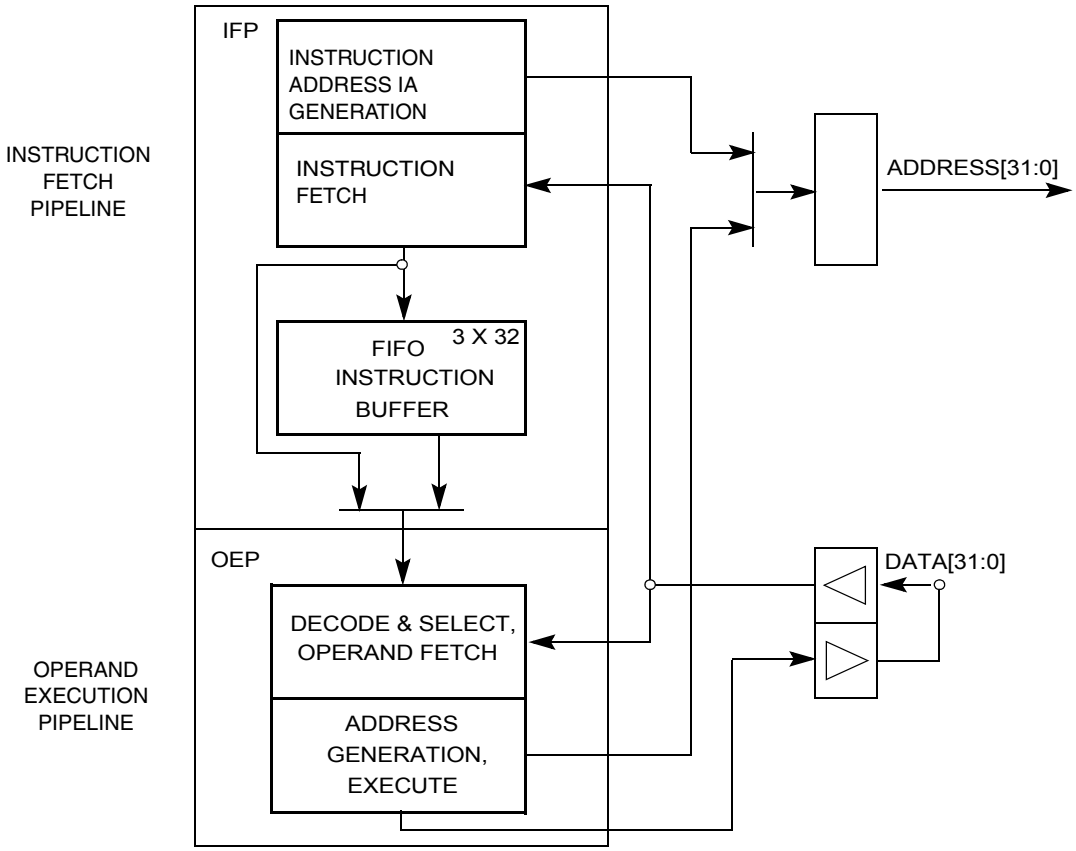


Figure 3-1. CF2 ColdFire Processor Core Pipelines

The processor core is comprised of two separate pipelines that are decoupled by an instruction buffer. The Instruction Fetch Pipeline (IFP) is responsible for instruction address generation and instruction fetch. The instruction buffer is a first-in-first-out (FIFO) buffer that holds prefetched instructions awaiting execution

in the Operand Execution Pipeline (OEP). The OEP includes two pipeline stages. The first stage decodes instructions and selects operands (DSOC); the second stage (AGEX) performs instruction execution and calculates operand effective addresses, if needed.

3.2 ColdFire Processor Memory Map and Register Definitions

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

3.2.1 User Memory Map and Register Description

Figure 3-2 shows the user Memory Map. The model is the same as the M68000 family of microprocessors and 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)

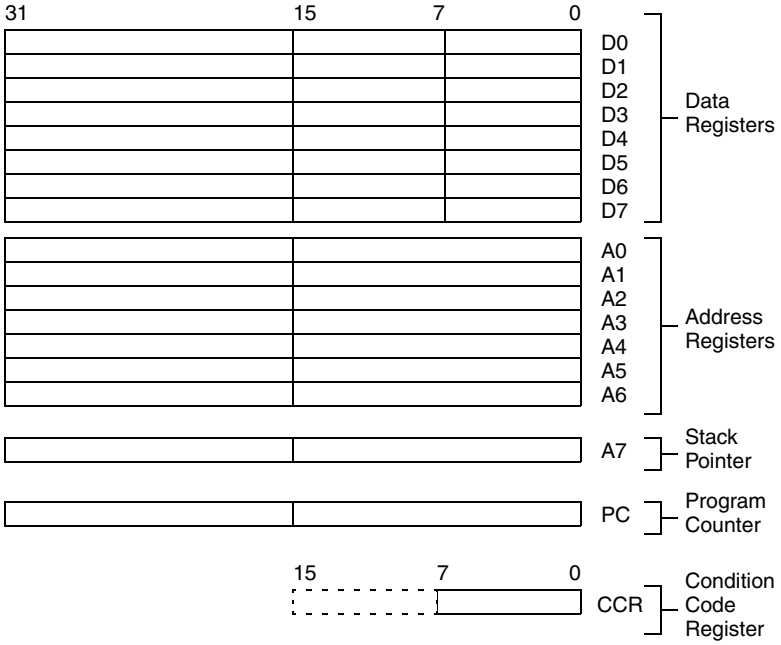


Figure 3-2. User Memory Map

3.2.1.1 Data Registers (D0–D7)

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

3.2.1.2 Address Registers (A0–A6)

Registers A0–A6 can be used as software stack pointers, index registers, or base address registers as well as for word and longword operations.

3.2.1.3 Stack Pointer (A7, SP)

The ColdFire architecture supports a single hardware stack pointer (A7) for explicit references as well as for implicit ones during stacking for subroutine calls and returns and exception handling. The initial value of A7 is loaded from the reset exception vector, address $0. The same register is used for both user and supervisor mode as well as word and longword operations.

A subroutine call saves the Program Counter (PC) on the stack and the return restores it from the stack. Both the PC and the Status Register (SR) are saved to the stack during the processing of exceptions and interrupts. The return from exception instruction restores the SR and PC values from the stack.

3.2.1.4 Program Counter (PC)

The PC contains the address of the next instruction to execute. During instruction execution and exception processing, the processor automatically increments the contents of the PC or places a new value in the PC, as appropriate. For some addressing modes, the PC can be used as a pointer for PC-relative operand addressing.

3.2.1.5 Condition Code Register (CCR)

The CCR is the least significant byte of the processor status register (SR). Refer to Status Register (SR) on Section 3.5, “Processor Exceptions” for more information. Bits 4–0 represent indicator flags based on results generated by processor operations. Bit 4, the extend bit (X bit), is also used as an input operand during multiprecision arithmetic computations.

Table 3-1. Condition Code Register (Bits 0–4)

7	6	5	4	3	2	1	0
–	–	–	X	N	Z	V	C

Table 3-2 describes the bits in the condition code register.

Table 3-2. CCR Register Field Description

Field	Code	Description
7–5	–	Reserved, should be cleared.
4	X	Extend condition code bit. Assigned the value of the carry bit for arithmetic operations; otherwise not affected or set to a specified result. Also used as an input operand for multiple-precision arithmetic.
3	N	Negative condition code bit. Set if the msb of the result is set; otherwise cleared.
2	Z	Zero condition code bit. Set if the result equals zero; otherwise cleared.

Table 3-2. CCR Register Field Description (continued)

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

3.2.2 Enhanced Multiply Accumulate Module (eMAC) User Memory Map and Register Description

The eMAC provides a variety of program-visible registers:

- Four 48-bit accumulators (Raccx = Racc0, Racc1, Racc2, Racc3)
- Eight 8-bit accumulator extensions (2 per accumulator), packaged as two 32-bit values for load and store operations (Raccext01, Raccext23)
- One 16-bit Mask Register (Rmask)
- One 32-bit Status Register (MACSR) including four indicator bits signaling product or accumulation overflow (one for each accumulator: PAV0, PAV1, PAV2, PAV3)

3.2.2.1 eMAC Instruction Set Summary

The eMAC unit supports the integer multiply operations defined by the baseline ColdFire architecture, as well as the multiply-accumulate instructions. Table 3-3 summarizes the eMAC unit instruction set.

Table 3-3. 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,Raccx MSAC Ry,RxSF,Raccx	Multiplies two operands, then adds/subtracts the product to/from an accumulator
Multiply Accumulate with Load	MAC Ry,RxSF,Rw,Raccx MSAC Ry,RxSF,Rw,Raccx	Multiplies two operands, then combines the product to an accumulator while loading a register with the memory operand
Load Accumulator	MOV.L {Ry,#imm},Raccx	Loads an accumulator with a 32-bit operand
Store Accumulator	MOV.L Raccx,Rx	Writes the contents of an accumulator to a CPU register
Copy Accumulator	MOV.L Raccy,Raccx	Copies a 48-bit accumulator
Load MAC Status Reg	MOV.L {Ry,#imm},MACSR	Writes a value to the MAC status register
Store MAC Status Reg	MOV.L MACSR,Rx	Write the contents of the MAC status register to a CPU register
Store MACSR to CCR	MOV.L MACSR,CCR	Write the contents of the MAC status register to the processor's CCR register
Load MAC Mask Reg	MOV.L {Ry,#imm},Rmask	Writes a value to the MAC Mask Register
Store MAC Mask Reg	MOV.L Rmask,Rx	Writes the contents of the MAC mask register to a CPU register

Table 3-3. eMAC Instruction Summary (continued)

Command	Mnemonic	Description
Load AccExtensions01	MOV.L {Ry,#imm},Raccext01	Loads the accumulator 0,1 extension bytes with a 32-bit operand
Load AccExtensions23	MOV.L {Ry,#imm},Raccext23	Loads the accumulator 2,3 extension bytes with a 32-bit operand
Store AccExtensions01	MOV.L Raccext01,Rx	Writes the contents of accumulator 0,1 extension bytes into a CPU register
Store AccExtensions23	MOV.L Raccext23,Rx	Writes the contents of accumulator 2,3 extension bytes into a CPU register

3.2.3 Supervisor Memory Map and Register Description

Only system programmers use the supervisor programming model to implement sensitive operating system functions, I/O control, and memory management. All accesses that affect the control features of ColdFire processors are in the supervisor memory map and register descriptions, which consists of the registers available to users as well as the following control registers:

- 16-bit status register (SR)
- 32-bit vector base register (VBR)

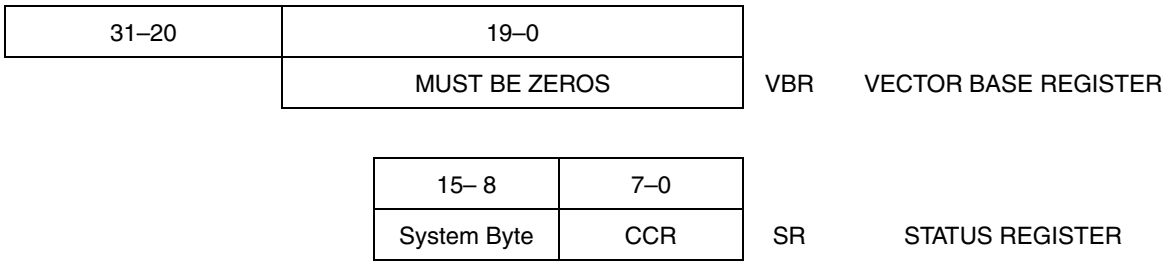


Figure 3-3. Supervisor Memory Map

Additional registers may be supported on a part-by-part basis.

The following sections describe the supervisor register descriptions.

3.2.3.1 Status Register (SR)

The SR stores the processor status and includes the CCR, the interrupt priority mask, and other control bits. In the supervisor mode, software can access the entire SR. In user mode, only the lower 8 bits are accessible (CCR). 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).

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

Figure 3-4. Status Register

Table 3-4. Status Register Field Descriptions

Field	Description
T	When set, the trace enable allows the processor to perform a trace exception after every instruction.
S	The supervisor / user state bit denotes whether the processor is in supervisor mode (S=1) or user mode (S=0).
M	The master / interrupt state bit is cleared by an interrupt exception, and can be set by software during execution of the RTE or move to SR instructions.
I [2–0]	The interrupt priority mask defines the current interrupt priority. Interrupt requests are inhibited for all priority levels less than or equal to the current priority, except the edge-sensitive level 7 request, which cannot be masked.

3.2.3.2 Vector Base Register (VBR)

The VBR contains the base address of the exception vector table in memory. The displacement of an exception vector is added to the value in this register to access the vector table. The lower 20 bits of the VBR are not implemented by ColdFire processors; they are zero, forcing the table to be aligned on a 1-megabyte boundary.

	30 –21	19 –0
Field	Exception vector table base address	–
Reset	0000_0000_0000_0000_0000_0000_0000_0000	
R/W	Written from a BDM serial command or from the CPU using the MOVEC instruction. VBR can be read from the debug module only. The upper 12 bits are returned, the low-order 20 bits are undefined.	
Rc [11–0]	0x801	

Figure 3-5. Vector Base Register (VBR)

3.3 Exception Processing Overview

Exception processing for ColdFire processors is streamlined for performance. The ColdFire processors provide a simplified exception processing model. The next section details the model.Differences from previous 68000 Family processors include:

- A simplified exception vector table
- Reduced relocation capabilities using the vector base register
- A single exception stack frame format
- Use of a single self-aligning system stack

ColdFire processors use an instruction restart exception model but do require more software support to recover from certain access errors. Refer to Section 3.5.1, “Access Error Exception,” for details.

Exception processing is comprised of four major steps and is defined as the time from the detection of the fault condition to the fetch of the first handler instruction has been initiated.

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 occurrence of an interrupt exception also forces the M bit to be cleared and the interrupt priority mask to be set to the level of the current interrupt request.

2. The processor determines the exception vector number. For all faults *except* interrupts, the processor performs this calculation based on the exception type. For interrupts, the processor performs an interrupt-acknowledge (IACK) bus cycle to obtain the vector number from a peripheral device. The IACK cycle is mapped to a special acknowledge 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 CF2 Core supports a single stack pointer in the A7 address register; therefore, there is no notion of separate supervisor or user stack pointers. As a result, the exception stack frame is created at a 0-modulo-4 address on the top of the current system stack. Additionally, 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-megabyte 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 x vector number). Once the exception vector has been fetched, the contents of the vector determine the address of the first instruction of the desired handler. After the instruction fetch for the first opcode of the handler has been initiated, exception processing terminates and normal instruction processing continues in the handler.

ColdFire 5200 processors support a 1024-byte vector table aligned on any 1-megabyte address boundary (see Table 3-5). The table contains 256 exception vectors where the first 64 are defined by Freescale and the remaining 192 are user-defined interrupt vectors.

The CF2 Core processor inhibits sampling for interrupts during the first instruction of all exception handlers. This allows any handler to effectively disable interrupts, if necessary, by raising the interrupt mask level contained in the status register.

Table 3-5. Exception Vector Assignments

Vector Numbers(s)	Vector Offset (HEX)	Stacked[1, 2] Program Counter	Assignment
0	$000	–	Initial stack pointer
1	$004	–	Initial program counter
2	$008	Fault	Access error
3	$00C	Fault	Address error
4	$010	Fault	Illegal instruction
5	$014	Fault	Divide by zero
6–7	$018-$01C	–	Reserved
8	$020	Fault	Privilege violation
9	$024	Next	Trace
10	$028	Fault	Unimplemented line-a opcode

Table 3-5. Exception Vector Assignments (continued)

Vector Numbers(s)	Vector Offset (HEX)	Stacked[1, 2] Program Counter	Assignment
11	$02C	Fault	Unimplemented line-f opcode
12	$030	Next	Debug interrupt
13	$034	–	Reserved
14	$038	Fault	Format error
15	$03C	Next	Uninitialized interrupt
16-23	$040-$05C	–	Reserved
24	$060	Next	Spurious interrupt
25–31	$064-$07C	Next	Level 1–7 autovectored interrupts
32–47	$080-$0BC	Next	Trap # 0–15 instructions
48–63	$0C0-$0FC	–	Reserved
64–255	$100-$3FC	Next	User-defined interrupts

1 "Fault" refers to the PC of the instruction that caused the exception.

2 "Next" refers to the PC of the next instruction that follows the instruction that caused the fault.

3.4 Exception Stack Frame Definition

The exception stack frame is shown in Figure 3-6. The first longword of the exception stack frame 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.

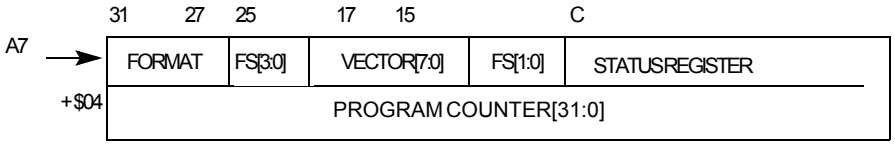


Figure 3-6. Exception Stack Frame Form

The 16-bit format/vector word contains 3 unique fields:

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

Table 3-6. Format Field Encoding

Original A7 @ Time of Exception, Bits 1:0	A7 @ 1st Instruction of Handler	Format Field
00	Original A7 - 8	4
01	Original A7 - 9	5
10	Original A7 - 10	6
11	Original A7 - 11	7

- 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 types of exceptions. See Table 3-7.

Table 3-7. Fault Status Encoding

FS[3:0]	Definition
00xx	Reserved
0100	Error on instruction fetch
0101	Reserved
011x	Reserved
1000	Error on operand write
1001	Attempted write to write-protected space
101x	Reserved
1100	Error on operand read
1101	Reserved
111x	Reserved

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

3.5 Processor Exceptions

3.5.1 Access Error Exception

The exact processor response to an access error depends on the type of 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 that occur during instruction prefetches that are then followed by a change of instruction flow do not generate an exception. When the processor attempts to execute an instruction with a faulted opword and/or extension words, the access error is signaled and the instruction aborted. For this type of exception, the programming model has not been altered by the instruction generating the access error.

If the access error occurs on an operand read, the processor immediately aborts the current instruction's execution and initiates exception processing. In this situation, any address register updates attributable to the auto-addressing modes, (e.g., (An)+,-(An)), have already been performed, so the programming model contains the updated An value. In addition, if an access error occurs during the execution of a MOVEM instruction loading from memory, any registers already updated *before* the fault occurs contains the operands from memory.

The 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.

3.5.2 Address Error Exception

Any attempted execution transferring control to an odd instruction address (i.e., 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 8 on an indexed effective addressing mode generates an address error as does an attempted execution of a full-format indexed addressing mode.

3.5.3 Illegal Instruction Exception

The MCF5251 processors decode the full 16-bit opcode and generate this exception if execution of an unsupported instruction is attempted. Additionally, attempting to execute an illegal line A or line F opcode generates unique exception types: vectors 10 and 11, respectively.

ColdFire processors do not provide illegal instruction detection on extension words of any instruction, including MOVEC. Attempting to execute an instruction with an illegal extension word causes undefined results.

3.5.4 Divide By Zero

Attempted division by zero causes an exception (vector 5, offset = 0x014) except when the PC points to the faulting instruction (DIVU, DIVS, REMU, REMS).

3.5.5 Privilege Violation

The attempted execution of a supervisor mode instruction while in user mode generates a privilege violation exception. Refer to the *ColdFire Programmer's Reference Manual* for lists of supervisor- and user-mode instructions.

3.5.6 Trace Exception

To aid in program development, the CF2 processors provide an instruction-by-instruction tracing capability. While in trace mode, indicated by the assertion of the T bit in the status register (SR[15] = 1), the completion of an instruction execution signals a trace exception. This functionality allows a debugger to monitor program execution.

The single exception to this definition is the STOP instruction. When the STOP opcode is executed, the processor core waits until an unmasked interrupt request is asserted, then aborts the pipeline and initiates interrupt exception processing.

Because ColdFire processors do not support hardware stacking of multiple exceptions, it is the responsibility of the operating system to check for trace mode after processing other exception types. For example, consider the execution of a TRAP instruction while in trace mode. The processor will initiate the TRAP exception and then pass 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[15] in the exception stack frame asserted) and pass control to the trace handler before returning from the original exception.

3.5.7 Debug Interrupt

This special type of program interrupt is covered in detail in Chapter 20, "Background Debug Mode (BDM) Interface." This exception 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).

3.5.8 RTE and Format Error Exceptions

When an RTE instruction is executed, the processor first examines the 4-bit format field to validate the frame type. For a ColdFire 5200 processor, any attempted execution of an RTE 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 68000 applications. On 680x0 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 5200 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.

3.5.9 TRAP Instruction Exceptions

Executing TRAP always forces an exception and is useful for implementing system calls. The trap instruction may be used to change from user to supervisor mode.

3.5.10 Interrupt Exception

The interrupt exception processing, with interrupt recognition and vector fetching, includes uninitialized and spurious interrupts as well as those where the requesting device supplies the 8-bit interrupt vector. Autovectoring may optionally be supported through the System Integration module (SIM).

3.5.11 Fault-on-Fault Halt

If a CF2 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 force the processor to exit this halted state.

3.5.12 Reset Exception

Asserting the reset input signal 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 S bit and disables tracing by clearing the T bit in the SR. This exception also clears the M bit and sets the processor’s interrupt priority mask in the SR to the highest level (level 7). Next, the VBR is initialized to zero ($00000000). 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 supervisor registers are also affected.
Refer to each of the modules in this manual for details on these registers.

After reset is negated, the core performs two longword read bus cycles. The first longword at address 0 is loaded into the stack pointer and the second longword at address 4 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 halted state.

3.6 Instruction Execution Timing

This section describes CF2 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 clock cycles. Each timing entry is presented as C(r/w) where:

- C—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—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.

3.6.1 Timing Assumptions

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

1. The operand execution pipeline (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 instruction fetch pipeline (IFP) to supply opwords and/or extension words.

2. The OEP does not experience any sequence-related pipeline stalls. For ColdFire 5200 processors, the most common example of this type 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 DSOC cycle of the store instruction. If a subsequent STORE instruction is encountered within this 2-cycle window, it will be stalled until the resource again becomes available. Thus, the maximum pipeline stall involving consecutive STORE operations is 2 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, i.e., 16 bit operands aligned on 0-modulo-2 addresses, 32 bit operands aligned on 0-modulo-4 addresses.

If the operand alignment fails these guidelines, it is misaligned. The processor core decomposes the misaligned operand reference into a series of aligned accesses as shown in Table 3-8.

Table 3-8. Misaligned Operand References

Address[1:0]	Size	KBUS Operations	Additional C(R/W)
X1	Word	Byte, Byte	2(1/0) if read 1(0/1) if write
X1	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

3.6.2 MOVE Instruction Execution Times

The execution times for the MOVE.{B,W} instructions are shown in Table 3-9, while Table 3-10 provides the timing 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.

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

Table 3-9. Move Byte and Word Execution Times

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d_{16},Ax)	(d_8,Ax,Xi)	(xxx).wl
Dn	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
An	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(An)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)

Table 3-9. Move Byte and Word Execution Times (continued)

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d_{16},Ax)	(d_8,Ax,Xi)	(xxx).wl
(An)+	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
-(An)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
(d_{16},An)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	–	–
(d_8,An,Xi)	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)	–	–	–
(d_{16},PC)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	–	–
(d_8,PC,Xi)	4(1/0)	4(1/1)	4(1/1)	4(1/1)	–	–	–
#<xxx>	1(0/0)	3(0/1)	3(0/1)	3(0/1)	–	–	–

Table 3-10. Move Long Execution Times

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d_{16},Ax)	(d_8,Ax,Xi)	(xxx).wl
Dn	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
An	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(An)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(An)+	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
-(An)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(d_{16},An)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	–	–
(d_8,An,Xi)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	–	–	–
(xxx).w	2(1/0)	2(1/1)	2(1/1)	2(1/1)	–	–	–
(xxx).l	2(1/0)	2(1/1)	2(1/1)	2(1/1)	–	–	–
(d_{16},PC)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	–	–
(d_8,PC,Xi)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	–	–	–
#<xxx>	1(0/0)	2(0/1)	2(0/1)	2(0/1)	–	–	–

3.7 Standard One Operand Instruction Execution Times

Table 3-11. One Operand Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
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)	–	–	–	–	–	–	–
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)

3.8 Standard Two Operand Instruction Execution Times

Table 3-12. Two Operand Instruction Execution Times (MACS)

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)

Table 3-12. Two Operand Instruction Execution Times (MACS) (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xn*SF) (d8,PC,Xn*SF)	xxx.wl	#xxx
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)	41/1)	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(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	–
btst	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	–	–	1(0/0)
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)
MAC.W	Ry,Rx	1(0/0)	–	–	–	–	–	–	–
MAC.L	Ry,Rx	1(0/0)	–	–	–	–	–	–	–
MSAC.W	Ry,Rx	1(0/0)	–	–	–	–	–	–	–
MSAC.L	Ry,Rx	3(0/0)	–	–	–	–	–	–	–
MAC.W	Ry,Rx,ea,Rw	–	2(1/0)	2(1/0)	2(1/0)	2(1/0)	–	–	–
MAC.L	Ry,Rx,ea,Rw	–	2(1/0)	2(1/0)	2(1/0)	2(1/0)	–	–	–
MSAC.W	Ry,Rx,ea,Rw	–	2(1/0)	2(1/0)	2(1/0)	2(1/0)	–	–	–
MSAC.L	Ry,Rx,ea,Rw	–	2(1/0)	2(1/0)	2(1/0)	2(1/0)	–	–	–
MOVEQ	#imm,Dx	–	–	–	–	–	–	–	1(0/0)
MULS.W	<ea>,Dx	4(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	12(1/0)	11(1/0)	9(0/0)
mulu.w	<ea>,Dx	4(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	12(1/0)	11(1/0)	9(0/0)
muls.l[1]	<ea>,Dx	≤ 4(0/0)	≤ 6(1/0)	≤ 6(1/0)	≤ 6(1/0)	≤ 6(1/0)	–	–	–

Table 3-12. Two Operand Instruction Execution Times (MACS) (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xn*SF) (d8,PC,Xn*SF)	xxx.wl	#xxx
mulu.l[1]	<ea>,Dx	≤ 4(0/0)	≤ 6(1/0)	≤ 6(1/0)	≤ 6(1/0)	≤ 6(1/0)	–	–	–
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)	–	–	–	–	–	–	–
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)
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)	35(1/0)	38(1/0)	38(1/0)	38(1/0)	–	–	–
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)	–	–	–	–	–	–	–

3.9 Miscellaneous Instruction Execution Times

Table 3-13. 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.w	CCR,Dx	1(0/0)	–	–	–	–	–	–	–
move.w	<ea>,CCR	1(0/0)	–	–	–	–	–	–	1(0/0)
move.w	SR,Dx	1(0/0)	–	–	–	–	–	–	–
move.w	<ea>,SR	7(0/0)	–	–	–	–	–	–	7(0/0) [2]
movec	Ry,Rc	9(0/1)	–	–	–	–	–	–	–
movem.l	<ea>,&list	–	1+n(n/0)	–	–	1+n(n/0)	–	–	–
movem.l	&list,<ea>	–	1+n(0/n)	–	–	1+n(0/n)	–	–	–
nop	–	3(0/0)	–	–	–	–	–	–	–
pea	<ea>	–	2(0/1)	–	–	2(0/1) [4]	3(0/1) [5]	2(0/1)	–
pulse	–	1(0/0)	–	–	–	–	–	–	–
stop	#imm	–	–	–	–	–	–	–	3(0/0) [3]
trap	#imm	–	–	–	–	–	–	–	15(1/2)
trapf	–	1(0/0)	–	–	–	–	–	–	–
trapf.w	–	1(0/0)	–	–	–	–	–	–	–

Table 3-13. Miscellaneous Instruction Execution Times (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
trapf.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)	3(1/0)
wdebug	<ea>	–	5(2/0)	–	–	5(2/0)	–	–	–

Notes:
n is the number of registers moved by the MOVEM opcode.
[1]≤ indicates that long multiplies have early termination after 9 cycles; thus, actual cycle count is operand independent
[2]If a MOVE.W #imm,SR instruction is executed and imm[13] = 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)

3.10 Branch Instruction Execution Times

Table 3-14. 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
BSR	–	–	–	–	–	3(0/1)	–	–	–
JMP	<ea>	–	3(0/0)	–	–	3(0/0)	4(0/0)	3(0/0)	–
JSR	<ea>	–	3(0/1)	–	–	3(0/1)	4(0/1)	3(0/1)	–
RTE	–	–	–	10(2/0)	–	–	–	–	–
RTS	–	–	–	5(1/0)	–	–	–	–	–

Table 3-15. BRA, Bcc Instruction Execution Times

Opcode	Forward Taken	Forward Not Taken	Backward Taken	Backward Not Taken
BRA	2(0/0)	–	2(0/0)	–
Bcc	3(0/0)	1(0/0)	2(0/0)	3(0/0)

Chapter 4
Phase-Locked Loop and Clock Dividers

This chapter provides detailed information about the operation and programming of the clock generation module as well as the recommended circuit settings. It also describes the audio clock generation and the system power states.

4.1 PLL Features

- The PLL locks to the crystal clock at the CRIN pin and produces a processor clock (PSTCLK) and a SYSCLK which is always 1/2 of the processor clock.
- The audio clock (AUDIOCLK) can be derived directly from CRIN or from the LRCK3/AUDIOCLK/GPIO43 input pin.
- The Audio DAC Master clocks MCLK1 and MCLK2 are derived directly from CRIN.
- The PLL is configured by writing to a configuration register.
- The PLL Configuration Register must always be programmed to Bypass mode before it is reprogrammed to change any clock frequency. In bypass mode, the crystal clock is fed to the processor (PSTCLK).
- When the PLL is switched from “bypass” to “normal operation”, the switch-over is delayed until the PLL is locked.
- The MCF5251 has a new block added to the output of the PLL / Clock Dividers to provide glitch-free Dynamic Clock Switching. This allows dynamic switching of the clock rate being fed to the CPU core and the system bus. This new block is controlled by a new 32-bit register called the ClockRate Register.

Figure 4-1 shows the PLL module and the frequency relationships of various clock signals.

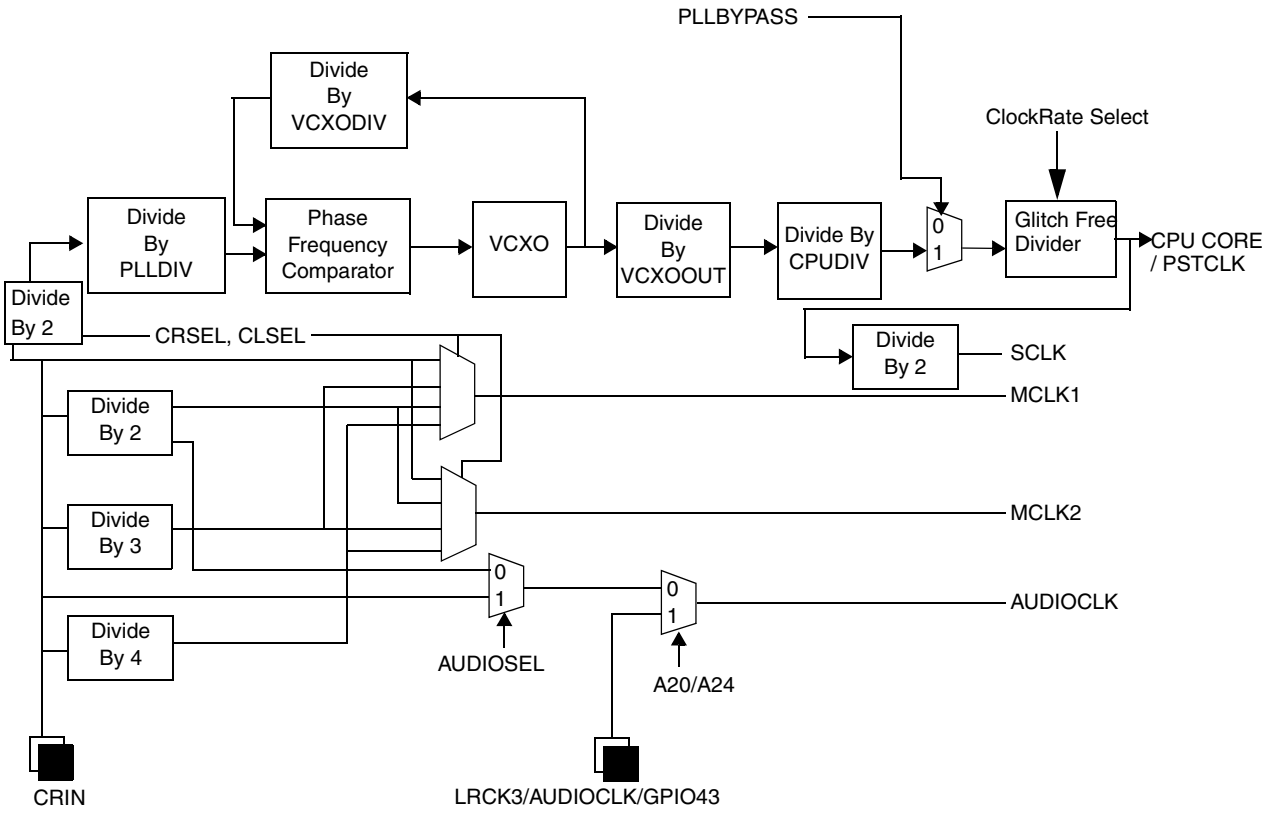


Figure 4-1. Phase-Locked Loop Module Block Diagram

4.2 PLL Memory Map and Register Definitions

The different settings for the PLL/clock module are summarized in Table 4-2.

Table 4-1. PLL Memory Map

Address MBAR2BAS +	Access	Size Bits	Name	Description	Reset
180	RW	32	PllConfig	PLL configuration register	0x02020088

Offset: MBAR2BAS + 0 x 180 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	LOCK	CLSEL				CPUDIV			CRSEL	AUDIO SEL			VCXODIV[7:4]			
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	VCXODIV[3:0]				SLEEP		PLL POWER DOWN	PLLDIV					VCXOOUT			PLL BYP
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-2. PLLCONFIG Register

Table 4-2. PLLCONFIG Field Descriptions

Field	Description	Notes
31 LOCK	Read-only bit. Set if PLL is locked	2
30–28 CLSEL	MCLK1 and MCLK2 select	7
27	Reserved.	–
26–24 CPUDIV	CPU clock divider	8, 9
23 CRSEL	0 Fin = CRIN 1 Fin = CRIN/2	3

22
AUDIOSEL
	If (pull-up on address pin A20/A24):
	Faudio = LRCK3/AUDIOCLK/GPIO43
	If (pull-down on address pin A20/A24 && AUDIOSEL = 1):
	Faudio = CRIN
	If (pull-down on address pin A20/A24 && AUDIOSEL = 0):
	Faudio = CRIN/2
		4, 12

21–20	Reserved, should be cleared.	–
19–12 VCXODIV	PLL compare frequency is VCXO frequency divided by (VCXODIV)	5, 10
11 SLEEP	0 Switch back to operational mode 1 Switch device to Sleep mode, including stopping clocks	11
10	Reserved, should be cleared.	–
9 PLLPWRDWN	0 PLL normal operation 1 Disable PLL to power-down mode	–
8–4 PLLDIV	Input frequency (Fin) is divided by (PllDiv) to determine the PLL compare frequency.	5
3–2 VCXOOUT	VCXO output divider	6

Table 4-2. PLLCONFIG Field Descriptions (continued)

Field	Description	Notes
1	Reserved, should be cleared.	–
0 PLLBYP	0 Bypass PLL and dividers 1 Switch to PLL after PLL is locked	1, 2

1. If this bit is 0, the PLL is by-passed, and CRIN is sent directly to the CPU and MCLKs. Always set the PLL to Bypass mode before changing any other bit in this register. Clock frequencies described in other notes are only valid when this bit is set 1.
2. PLL may require up to 10 mS to lock
3. Fin is input frequency to PLL. Nominal setting for CRsel is '1' for 33.8688 MHz X-tal, '0' for 16.9344 MHz or 11.2896MHz X-tal.
4. Faudio is clock for audio interfaces. Typically 11.2896 or 16.9344 or 22.579 or 33.8688 MHz.
5. Fvcxo = Fin × (2 × VCXODIV)/ (PLLDIV)
6. FVCXOOut depends on Fvcxo (note 5) and vcxoout setting as shown in Table 4-3.

Table 4-3. Vcxoout Settings

Vcxoout Setting	FVCXOOut
0	Don't use
1	Fvcxo
2	Fvcxo/2
3	Don't use

7. Field determines frequency output on MCLK1 and MCLK2 pins

Table 4-4. Crsel and CLsel Settings

Crsel	CLsel	Frequency MCLK1	Frequency MCLK2
0	000	CRIN	CRIN/2
0	001	CRIN	CRIN
0	010	CRIN/2	CRIN/2
0	011	CRIN/2	CRIN
1	000	CRIN/2	CRIN/2
1	001	CRIN/2	CRIN/3
1	010	CRIN/2	CRIN/4
1	011	CRIN/3	CRIN/2
1	100	CRIN/3	CRIN/3
1	101	CRIN/3	CRIN/4
1	110	CRIN/4	CRIN/2
1	111	CRIN/4	CRIN/3

When frequency is CRIN/2 or CRIN/4, duty cycle is 50%. When frequency is CRIN/3, duty cycle is 33%.

8. Fcpu = FVCXOOUT / CPUDIV; Fcpu is the frequency the processor is running at.

9. If field is “000”, divide by 8
10. Fvcxo min. 200 MHz max. is 400 MHz
11. This bit controls power-down.
12. Faudio is the audio clock. It is LRCK3/AUDIOCLK/GPIO43 when address pin A20/A24 is connected with a pull-up to Vdd, it is CRIN or CRIN/2 when address pin A20/A24 is connected with a pull-down to GND.

4.2.1 PLL Operation

The input to the PLL is either CRIN, or CRIN divided by two. Selection is done by CRSEL. The PLL divides this input frequency by a programmable division factor *(PLLDIV)*. In the PLL phase/frequency detector, this divided clock is compared with the VCXO output clock divided by *(VCXODIV)*. As a result, $Fvcxo = Fin \times (2 \times VCXODIV) / (PLLDIV)$.

NOTE

The PLL lock counter is designed for worst case audio input frequency (Fin) of 33.8688 MHz. This will result in the required 0.5 ms for the PLL to lock. Other Fin frequencies can be used, however, the resulting lock time will be slightly longer.

In a second step, this VCXO clock is divided by *(VCXOOUT * CPUDIV)* to create the CPU clock *PSTCLK*.

The multiplexers that switch between PLL clock and CRIN is glitch-free, so no system reset is needed when switching between bypass and PLL modes.

NOTE

It is important that before reprogramming the PLL division factors, users must switch to PLL bypass mode. After reprogramming, users may immediately switch back to PLL enabled mode. Switching back is delayed internally until the PLL is locked.

4.2.2 PLL Lock-In Time

PLL lock time is typically around 10 ms.

4.2.3 PLL Electrical Limits

Due to implementation of the block, some limits apply to the PLL block. These limitations are shown in Table 4-5.

Table 4-5. PLL Electrical Limits

Name	Minimum Frequency MHz	Maximum Frequency MHz	Reason
Fvcxo	200	400	PLL limitations
Fcpu	0	140	Maximum operating frequency of device
Fin/PLLDIV	2	8	PLL limitations

4.3 Dynamic Clock Switching

The Glitch-free Clock Rate divider block works on the PDM (Pulse Density Modulation) principal. As it is a divider, it only reduces the clock rate output from the PLL module. It does this by gating out (removing) clock pulses to reach the desired clock rate (operating frequency). This implies that clock pulses after the Clock Rate divider block may not be equi-distant depending on the chosen divider value.

The re-action time of the Clock Rate divider block is in the order of 20nS.

The Clock Rate divider block is controlled by a 32-bit register as shown in Table 4-3

Offset: MBAR2BAS + 0x170 Access: User read/write

	31	30	29	28	27	26	25	24	23	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	Clock rate select							
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 4-3. ClockRate Register

Table 4-6. ClockRate Field Descriptions

Field	Description
31–8	Reserved, should be cleared.
7–0 Clock Rate Select	Sets the divider to pass-thru mode and no clock rate reduction is applied. At reset, this bit field is set to 0xFF. Clock Rate Select = round ((Desired Clock Rate / PLL Output Frequency) × 255)

NOTE

Do not make the PLL output frequency any higher than required for the actual application. The higher the PLL operating frequency then the higher the power consumption of the PLL block.

4.4 Audio Clock Generation

The audio clocks and output DAC clocks are derived directly from the CRIN pin. Clock settings depend on CRSEL, CLSEL, and AUDIOSEL bits, as explained in Table 4-7. As the table shows, the AUDIO_CLOCK is completely derived from the AUDIOSEL bit, and this clock is independent of the other select bits. For the DAC clocks (MCLK2 and MCLK1) the relationship between CRSEL and CLSEL is defined in Table 4-7.

Table 4-7. PLLCR Bit Fields

PLLCR[CLSEL] (Bits 30–28)	PllCR CRsel (Bit 23)	pllCR Config Audiosel (Bit 22)	AUDIO_CLOCK	MCLK2	MCLK1
000	1	1	CRIN	CRIN	CRIN/2
001	1	1	CRIN	CRIN	CRIN
010	1	1	CRIN	CRIN/2	CRIN/2

Table 4-7. PLLCR Bit Fields (continued)

PLLCR[CLSEL] (Bits 30–28)	PllCR CRsel (Bit 23)	pllCR Config Audiosel (Bit 22)	AUDIO_CLOCK	MCLK2	MCLK1
011	1	1	CRIN	CRIN/2	CRIN
100	1	1	CRIN	CRIN	CRIN/2
101	1	1	CRIN	CRIN	CRIN
110	1	1	CRIN	CRIN/2	CRIN/2
111	1	1	CRIN	CRIN/2	CRIN
000	1	0	CRIN/2	CRIN	CRIN/2
001	1	0	CRIN/2	CRIN	CRIN
010	1	0	CRIN/2	CRIN/2	CRIN/2
011	1	0	CRIN/2	CRIN/2	CRIN
100	1	0	CRIN/2	CRIN	CRIN/2
101	1	0	CRIN/2	CRIN	CRIN
110	1	0	CRIN/2	CRIN/2	CRIN/2
111	1	0	CRIN/2	CRIN/2	CRIN

Note: MCLK1 will output a clock signal just after reset, it can be configured as GPIO if so desired. The frequency of the clock will be the same as CRIN prior to initialization of the PLL.

Note: The AUDIO_CLOCK can also be derived from the LRCK3/AUDIOCLK/GPIO43 pin.

The multiplexer that switches AUDIO_CLOCK between CRIN and CRIN/2 is glitch free. No reset is needed after switching audio clock. For the MCLK1 and MCLK2 clocks, the divide by 2 is 50% duty cycle, divide by 3 is 33% duty cycle, and divide by 4 is 50% duty cycle.

4.5 Reduced Power Mode

To save power, it is recommended that users reduce the frequency of the CPU clocks. This is done by reprogramming the PLLCONFIG register.

The PLL is also configured with a power down bit. This bit, when set to 1, this sets the PLL to Sleep mode. In Sleep mode, the VCXO is turned off.

NOTE

The PLL must go through the re-locking procedure when it is re-enabled.

4.6 Sleep / Wake-up Mode

The device can be put in a low power Sleep mode, where all internal clocks and all on-chip functions are stopped. In Sleep mode, the only block still functional is the on-chip voltage regulator. All the other analog features are put in to low-power operation and all digital functions are stopped.

4.6.1 Enter Sleep Mode

To enter Sleep mode, first put the PLL in bypass mode by clearing bit 0 of PLLCONFIG. Next, switch off all activity by setting the SLEEPMODE bit (bit 11) in PLLCONFIG. As a result, the device will go in to a low-power standby mode. All clocks are stopped.

4.6.2 Exit Sleep Mode

To exit Sleep mode, apply a LOW level to the $\overline{\text{WAKEUP}}$/GPIO21 input pin. This will power up all the analog functions, and restart the on-chip clocks after a 10 ms time delay. Program code should then clear bit 11 in PLLCONFIG before the low level on the $\overline{\text{WAKEUP}}$/GPIO21 pin is removed.

If $\overline{\text{WAKEUP}}$/GPIO21 pin is programmed as its GPO21 function, it may be impossible to exit Sleep mode by applying low level to the $\overline{\text{WAKEUP}}$ pin.
To exit Sleep mode under this circumstance ensure the $\overline{\text{WAKEUP}}$/GPIO21 pin is in its non-GPIO function prior to entering Sleep mode.

4.7 Selecting Audio_clock Input

During power-on reset, the value on pin A20/A24 is sensed. A 10 Kohm resistor should be connected between these pins and VDD/GND. Depending whether a pull-up or pull-down is mounted, different options are selected.

Table 4-8. Audio_Clock Selection Fields

Pin	Description
A20/A24	Pull-down: Audio clock taken from CRIN Pull-up: Audio clock taken from LRCK3/AUDIOCLK/GPIO43 pin

4.8 Recommended Settings

Many valid PLL settings exist. However, in many cases some limitations apply so that only a few typical settings will be used. In a typical system, the following limitations may exist:

- Users may want to run the processor at 120, 96, or 72 MHz clock frequency
- MCLK1 may be 11.2896 MHz (a typical value) see Table 4-7 in this section for further definition.

As a result, the user may select a 11.2896 MHz X-TAL as the CRIN and use the settings shown in Table 4-9.

Table 4-9. Recommended PLL Settings

X-Tal Freq MHz	CPU Div	CRSel	Vcxo Div	Pll Div	Vcxo Out	CPU Clock MHz
11.2896	2	0	42	4	1	120
11.2896	3	0	51	4	1	96
11.2896	4	0	51	4	1	72

Chapter 5
Instruction Cache

This chapter describes the physical organization, operation, memory map and register descriptions for the MCF5251 instruction cache.

5.1 Instruction Cache Features

- 8 Kbyte Direct-Mapped Cache
- Single-Cycle Access on Cache Hits
- Physically Located on the ColdFire® Core High-Speed Local Bus
- Nonblocking Design to Maximize Performance
- 16 Byte Line-Fill Buffer
- Configurable Cache Miss-Fetch Algorithm

5.2 Block Diagram

Figure 5-1 illustrates the block diagram for the instruction cache memory.

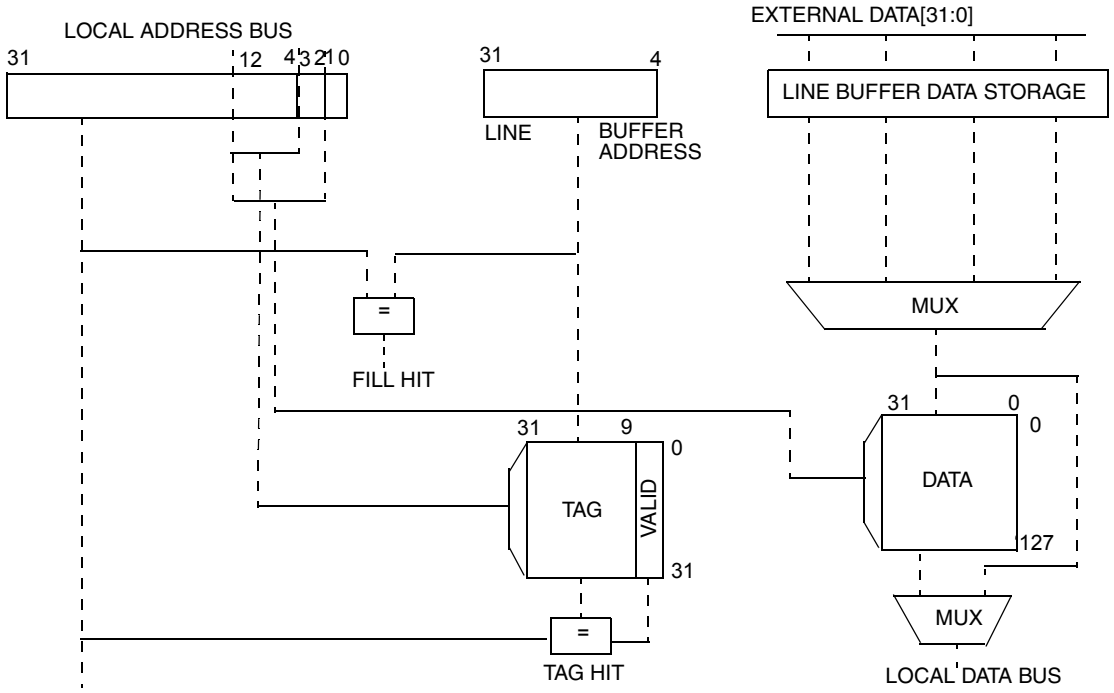


Figure 5-1. Instruction Cache Block Diagram

5.3 Instruction Cache Physical Organization

The instruction cache is a direct-mapped single-cycle memory, organized as 512 lines, each containing 16 Bytes. The memory storage consists of a 512-entry tag array (containing addresses and a valid bit), and the data array containing 8 Kbyte of instruction data, organized as 2048 × 32 bits.

The two memory arrays are accessed in parallel: bits [12:4] of the instruction fetch address provide the index into the tag array, and bits [12:2] addressing the data array. The tag array outputs the address mapped to the given cache location along with the valid bit for the line. This address field is compared to bits [31:12] of the instruction fetch address from the local bus to determine if a cache hit in the memory array has occurred. If the desired address is mapped into the cache memory, the output of the data array is driven onto the ColdFire core's local data bus completing the access in a single cycle.

The tag array maintains a single valid bit per line entry. Accordingly, only entire 16 byte lines are loaded into the instruction cache.

The instruction cache also contains a 16 byte fill buffer that provides temporary storage for the last line fetched in response to a cache miss. With each instruction fetch, the contents of the line fill buffer are examined. Thus, each instruction fetch address examines both the tag memory array and the line fill buffer to see if the desired address is mapped into either hardware resource. A cache hit in either the memory array or the line-fill buffer is serviced in a single cycle. Because the line fill buffer maintains valid bits on a longword basis, hits in the buffer can be serviced immediately without waiting for the entire line to be fetched.

If the referenced address is not contained in the memory array or the line-fill buffer, the instruction cache initiates the required external fetch operation. In most situations, this is a 16 byte line-sized burst reference.

The hardware implementation is a nonblocking design, meaning the ColdFire core's local bus is released after the initial access of a miss. Thus, the cache or the SRAM module can service subsequent requests while the remainder of the line is being fetched and loaded into the fill buffer.

5.4 Instruction Cache Operation

The instruction cache is physically connected to the ColdFire core local bus, allowing it to service all instruction fetches from the ColdFire core and certain memory fetches initiated by the debug module. Typically, the debug module's memory references appear as supervisor data accesses but the unit can be programmed to generate user-mode accesses and/or instruction fetches. The instruction cache processes any instruction fetch access in the normal manner.

5.4.1 Interaction with Other Modules

Because both the instruction cache and high-speed SRAM module are connected to the ColdFire core local data bus, certain user-defined configurations can result in simultaneous instruction fetch processing.

If the referenced address is mapped into the SRAM module, that module will service the request in a single cycle. In this case, data accessed from the instruction cache is simply discarded and no external memory references are generated. If the address is not mapped into the SRAM space, the instruction cache handles the request in the normal fashion.

5.4.2 Memory Reference Attributes

For every memory reference the ColdFire core or the debug module generates, a set of "effective attributes" is determined based on the address and the Access Control Registers (ACR0, ACR1). This set of attributes includes the cacheable/noncacheable definition, the precise/imprecise handling of operand write, and the write-protect capability.

In particular, each address is compared to the values programmed in the Access Control Registers (ACR). If the address matches one of the ACR values, the access attributes from that ACR are applied to the reference. If the address does not match either ACR, then the default value defined in the Cache Control Register (CACR) is used. The specific algorithm is as follows:

```
if (address = ACR0_address including mask)
Effective Attributes = ACR0 attributes
else if (address = ACR1_address including mask)
Effective Attributes = ACR1 attributes
else Effective Attributes = CACR default attributes
```

5.4.3 Cache Coherency and Invalidation

The instruction cache does not monitor ColdFire core data references for accesses to cached instructions. Therefore, software must maintain cache coherency by invalidating the appropriate cache entries after modifying code segments.

The cache invalidation can be performed in two ways. The assertion of bit 24 in the CACR forces the entire instruction cache to be marked as invalid. The invalidation operation requires 512 cycles because the cache sequences through the entire tag array, clearing a single location each cycle. Any subsequent instruction fetch accesses are postponed until the invalidation sequence is complete.

The privileged CPUSHL instruction can invalidate a single cache line. When this instruction is executed, the cache entry defined by bits[12:4] of the source address register is invalidated, provided bit 28 of the CACR is cleared.

These invalidation operations can be initiated from the ColdFire core or the debug module.

5.4.4 Reset

A hardware reset clears the CACR disabling the instruction cache. The contents of the tag array are not affected by the reset. Accordingly, the system startup code must explicitly perform a cache invalidation by setting CACR[24] before the cache can be enabled.

5.4.5 Cache Miss Fetch Algorithm/Line Fills

As detailed in Section 5.1, "Instruction Cache Features," the instruction cache hardware includes a 16-byte line fill buffer for providing temporary storage for the last fetched instruction.

With the cache enabled as defined by CACR[31], a cacheable instruction fetch that misses in both the tag memory and the line-fill buffer generates a external fetch. The size of the external fetch is determined by

the value contained in the 2-bit CLNF field of the CACR and the miss address. Table 5-1 shows the relationship between the CLNF bits, the miss address, and the size of the external fetch.

Depending on the runtime characteristics of the application and the memory response speed, overall performance may be increased by programming the CLNF bits to values {00, 01}.

Table 5-1. Initial Fetch Offset versus CLNF Bits

CLNF[1:0]	Longword Address Bits			
	00	01	10	11
00	Line	Line	Line	Longword
01	Line	Line	Longword	Longword
1X	Line	Line	Line	Line

For all cases of a line-sized fetch, the critical longword defined by bits [3:2] of the miss address is accessed first followed by the remaining three longwords that are accessed by incrementing the longword address in a modulo-16 fashion is shown in the following example code:

```
if miss address[3:2] = 00
fetch sequence = {$0, $4, $8, $C}
if miss address[3:2] = 01
fetch sequence = {$4, $8, $C, $0}
if miss address[3:2] = 10
fetch sequence = {$8, $C, $0, $4}
if miss address[3:2] = 11
fetch sequence = {$C, $0, $4, $8}
```

Once an external fetch has been initiated and the data loaded into the line-fill buffer, the instruction cache maintains a special “most-recently-used” indicator that tracks the contents of the fill buffer versus its corresponding cache location. At the time of the miss, the hardware indicator is set, marking the fill buffer as “most recently used.” If a subsequent access occurs to the cache location defined by bits [8:4] of the fill buffer address, the data in the cache memory array is now most recently used, so the hardware indicator is cleared. In all cases, the indicator defines whether the contents of the line fill buffer or the memory data array are most recently used. At the time of the next cache miss, the contents of the line-fill buffer are written into the memory array if the entire line is present, and the fill buffer data is still most recently used compared to the memory array.

The fill buffer can also be used as temporary storage for line-sized bursts of non-cacheable references under control of CACR[10]. With this bit set, a noncacheable instruction fetch is processed as defined by Table 5-2. For this condition, the fill buffer is loaded and subsequent references can hit in the buffer, but the data is never loaded into the memory array.

Table 5-2 shows the relationship between CACR bits 31 and 10 and the type of instruction fetch.

Table 5-2. Instruction Cache Operation as Defined by CACR [31,10]

CACR[31]	CACR[10]	Type of Instruction Fetch	Description
0	0	N/A	Instruction cache is completely disabled; all fetches are word, longword in size
0	1	N/A	All fetches are word, longword in size
1	X	Cacheable	Fetch size is defined by Table 5-1 and contents of the line-fill buffer can be written into the memory array
1	0	Noncacheable	All fetches are longword in size, and not loaded into the line-fill buffer
1	1	Noncacheable	Fetch size is defined by Table 5-1 and loaded into the line-fill buffer, but are never written into the memory array

5.5 Instruction Cache Memory Map and Register Definitions

Three supervisor registers define the operation of the instruction cache and local bus controller: the Cache Control Register (CACR) and two Access Control Registers (ACR0, ACR1).

5.5.1 Instruction Cache Registers Memory Map

Table 5-3 shows the memory map of the Instruction cache and access control registers.

The following list describes several key issues regarding the memory map table:

- The Cache Control Register and Access Control Registers can only be accessed in supervisor mode using the MOVEC instruction with an Rc value of $002, $004 and $005, respectively.
- Addresses not assigned to the registers and undefined register bits are reserved for future expansion. Write accesses to these reserved address spaces and reserved register bits have no effect; read accesses will return zeros.
- The reset value column indicates the initial value of the register at reset. Certain registers may contain random values after reset.

The access column indicates if the corresponding register allows both read/write functionality (R/W), read-only functionality (R), or write-only functionality (W). If a read access to a write-only register is attempted, zeros will be returned. If a write access to a read-only register is attempted the access will be ignored and no write will occur.

Table 5-3. Memory Map of I-Cache Registers

Address	Name	Width	Description	Reset Value	Access
MOVEC with $002	CACR	32	Cache Control Register	$0000	W
MOVEC with $004	ACR0	32	Access Control Register 0	$0000	W
MOVEC with $005	ACR1	32	Access Control Register 1	$0000	W

5.5.2 Instruction Cache Register

5.5.2.1 Cache Control Register

The CACR controls the operation of the instruction cache. The CACR provides a set of default memory access attributes used when a reference address does not map into the spaces defined by the ACRs.

The CACR is a 32-bit write-only supervisor control register. It is accessed in the CPU address space using the MOVEC instruction with an Rc encoding of $002. The CACR can be read when in Background Debug mode (BDM). At system reset, the entire register is cleared.

Address MOVEC with $002 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CENB			CPDI	CFRZ			CINV								
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						CEIB	DCM	DBWE		DWP					CLNF1	CLNF2
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 5-4. Cache Control Register Field Descriptions

Bit Name	Description
31 CENB	The Cache Enable bit generally provides longword references used for sequential fetches. If the processor branches to an odd word address, a word-sized fetch is generated. The memory array of the instruction cache is enabled only if CENB is asserted. 0 Cache disabled 1 Cache enabled
30–29	Reserved, should be cleared.
28 CPDI	When the disable CPUSHL Invalidation instruction is executed, the cache entry defined by bits [8:4] of the address is invalidated if CPDI = 0. If CPDI = 1, no operation is performed. 0 Enable invalidation 1 Disable invalidation
27 CFRZ	The Cache Freeze bit allows users to freeze the contents of the cache. When CFRZ is asserted line fetches can be initiated and loaded into the line-fill buffer, but a valid cache entry can not be overwritten. If a given cache location is invalid, the contents of the line-fill buffer can be written into the memory array while CFRZ is asserted. 0 Normal operation 1 Freeze valid cache lines
26–25	Reserved, should be cleared.

Table 5-4. Cache Control Register Field Descriptions (continued)

Bit Name	Description
24 CINV	The Cache Invalidate bit forces the cache to invalidate each tag array entry. The invalidation process requires 32 machine cycles, with a single cache entry cleared per machine cycle. The state of this bit is always read as a zero. After a hardware reset, the cache must be invalidated before it is enabled. 0 No operation 1 Invalidate all cache locations
23–11	Reserved, should be cleared.
10 CEIB	The Cache Enable Noncacheable Instruction Bursting bit enables the line-fill buffer to be loaded with burst transfers under control of CLINF[1:0] for non-cacheable accesses. Noncacheable accesses are never written into the memory array. 0 Disable burst fetches on noncacheable accesses 1 Enable burst fetches on noncacheable accesses
9 DCM	The Default Cache Mode bit defines the default cache mode: 0 is cacheable, 1 is noncacheable. For more information on the selection of the effective memory attributes, see Section 5.4.2, “Memory Reference Attributes.” 0 Default cacheable 1 Default noncacheable
8 DBWE	The Default Buffered Write Enable bit defines the default value for enabling buffered writes. If DBWE = 0, the termination of an operand write cycle on the processor's local bus is delayed until the external bus cycle is completed. If DBWE = 1, the write cycle on the local bus is terminated immediately and the operation buffered in the bus controller. In this mode, operand write cycles are effectively decoupled between the processor's local bus and the external bus. Generally, enabled buffered writes provide higher system performance but recovery from access errors can be more difficult. For the ColdFire CPU, reporting access errors on operand writes is always imprecise and enabling buffered writes simply further decouples the write instruction from the signaling of the fault. 0 Disable buffered writes 1 Enable buffered writes
7	Reserved, should be cleared.
6 DWP	Default Write Protection 0 Read and write accesses permitted 1 Only read accesses permitted
5–2	Reserved, should be cleared.
1–0 CLNF	The Cache Line Fill bits control the size of the memory request the cache issues to the bus controller for different initial line access offsets. Table 5-5 shows the fetch size.

Table 5-5. External Fetch Size Based on Miss Address and CLNF

CLNF[1:0]	Longword Address Bits			
	00	01	10	11
00	Line	Line	Line	Longword
01	Line	Line	Longword	Longword
10	Line	Line	Line	Line
11	Line	Line	Line	Line

5.5.2.2 Access Control Registers

The access control registers ACR0 and ACR1, provide a definition of memory reference attributes for two memory regions (one per ACR). This set of effective attributes is defined for every memory reference using the ACRs or the set of default attributes contained in the CACR. The ACRs are examined for every memory reference that is NOT mapped to the SRAM module.

The ACRs are 32-bit write-only supervisor control registers. They are accessed in the CPU address space using the MOVEC instruction with an Rc encoding of $004 and $005. The ACRs can be read when in background debug mode (BDM). At system reset, the registers are cleared.

Address MOVEC with $004
MOVEC with $005

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	BA31	BA30	BA29	BA28	BA27	BA26	BA25	BA24	BAM31	BAM30	BAM29	BAM28	BAM27	BAM26	BAM25	BAM24
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	EN	SM1	SM0							CM	BWE			WP		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-3. Access Control Registers (ACR0, ACR1)

Table 5-6. Access Control Registers Field Descriptions

Bit Name	Description
31–24 BA	The Base Address. An 8-bit field compared to the address bits [31:24] from the processor's local bus under control of the ACR address mask. If the address matches, the attributes for the memory reference are sourced from the given ACR.
23–16 BAM	The Base Address Mask [31:24] 8-bit field can mask any bit of the AB field comparison. If a bit in the AM field is set, then the corresponding bit of the address field comparison is ignored.
15 EN	The Enable bit defines the ACR enable. Hardware reset clears this bit, disabling the ACR. 0 ACR disabled 1 ACR enabled
14–13 SM1, SM0	The Supervisor mode two-bit field allows the given ACR to be applied to references based on operating privilege mode of the ColdFire processor. The field uses the ACR for user references only, supervisor references only, or all accesses. 00 Match if user mode 01 Match if supervisor mode 1x Match always. Ignore user/supervisor mode
12–7	Reserved, should be cleared.
6 CM	The Cache Mode bit defines the cache mode: 0 is cacheable, 1 is noncacheable. 0 Caching enabled 1 Caching disabled

Table 5-6. Access Control Registers Field Descriptions (continued)

Bit Name	Description
5 BWE	The Buffered Write Enable bit defines the value for enabling buffered writes. If BWE = 0, the termination of an operand write cycle on the processor's local bus is delayed until the external bus cycle is completed. If BWE = 1, the write cycle on the local bus is terminated immediately and the operation is then buffered in the bus controller. In this mode, operand write cycles are effectively decoupled between the processor's local bus and the external bus. Generally, the enabling of buffered writes provides higher system performance but recovery from access errors may be more difficult. For the ColdFire CPU, the reporting of access errors on operand writes is always imprecise, and enabling buffered writes simply decouples the write instruction from the signaling of the fault even more. 0 Don't buffer writes 1 Buffer writes
4–3	Reserved, should be cleared.
2 WP	The Write Protect bit defines the write-protection attribute. If the effective memory attributes for a given access select the WP bit, an access error terminates any attempted write with this bit set. 0 Read and write accesses permitted 1 Only read accesses permitted
1–0	Reserved, should be cleared.

Chapter 6
Static RAM (SRAM)

This chapter describes the SRAM operation, memory map, register descriptions, initialization and SRAM power management.

6.1 SRAM Features

- Two 64 Kbyte SRAMS
- Single-cycle access
- Physically located on processor's high-speed local bus
- Memory location programmable on any 64 Kbyte address boundary
- Byte, word, longword address capabilities

6.2 SRAM Operation

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 modulo-64K address within the 4GB address space. 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.

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

Only SRAM1 can be accessed by the DMA controller of the MCF5251. SRAM0 and SRAM1 are made up of two memory arrays each consisting of 2048 lines, with 16 Bytes in each line.

As SRAM1 can be accessed by the DMA then the split in the array (Upper 32K bank and Lower 32K bank) allows simultaneous access by both DMA and the CPU. Figure 1-1 shows this concept.

6.3 SRAM Memory Map and Register Definitions

The SRAM programming model includes a description of the SRAM base address register (RAMBAR), SRAM initialization, and power management.

6.3.1 SRAM Base Address Register

The configuration information in the SRAM Base Address Register (RAMBAR[0:1]) controls the operation of the SRAM module.

- There are 2 RAMBAR registers. One for SRAM0, the second for SRAM1.

- The RAMBAR register holds the base address of the SRAM. The MOVEC instruction provides write-only access to this register.
- The RAMBAR registers can be read or written from the Debug module in a similar manner.
- 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.
- The RAMBAR valid bit is cleared by reset, disabling the SRAM module. All other bits are unaffected.

The RAMBAR register contains several control fields. These fields are detailed in the following tables.

NOTE

All unused bits in the RAMBAR register must be initialized to zero

Address CPU + $C04 — Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BA31	BA30	BA29	BA28	BA27	BA26	BA25	BA24	BA23	BA22	BA21	BA20	BA19	BA18	BA17	BA16
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BA15	BA14						WP			C/I	SC	SD	UC	UD	V
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0

Figure 6-1. SRAM Base Address Register (RAMBAR0)

Address CPU + $C05 — Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BA31	BA30	BA29	BA28	BA27	BA26	BA25	BA24	BA23	BA22	BA21	BA20	BA19	BA18	BA17	BA16
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BA15	BA14			PRI1	PRI2	SPV	WP			C/I	SC	SD	UC	UD	V
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0

Figure 6-2. SRAM1 Base Address Register (RAMBAR1)

Table 6-1. SRAM(1) Base Address Register (RAMBAR*n*) Field Descriptions

<table>
<tr><th>Bit Name</th><th>Description</th></tr>
<tr><td>31–14
BA</td><td>The Base Address field defines the modulo-64K base address of the SRAM module. The SRAM memory occupies a 64 Kbyte space defined by the contents of the Base Address field. By programming this field, the SRAM may be located on any 64 Kbyte boundary within the processor's four gigabyte address space.</td></tr>
<tr><td>13–12
(SRAM1)
13–9
(SRAM)</td><td>Reserved, should be cleared.</td></tr>
<tr><td>11–10
PRI1, PRI2
(SRAM1 Only)</td><td>The PRI1 priority bit determines if DMA or CPU has priority in upper 32k bank of memory. PRI2 determines if DMA or CPU has priority in lower 32k bank of memory. If bit is set, DMA has priority. If bit is reset, CPU has priority. Priority is determined by the following table:
<table>
<tr><th>PRI[1:2]</th><th>Upper Bank Priority</th><th>Lower Bank Priority</th></tr>
<tr><td>2'b00</td><td>CPU Accesses</td><td>CPU Accesses</td></tr>
<tr><td>2'b01</td><td>CPU Accesses</td><td>DMA Accesses</td></tr>
<tr><td>2'b10</td><td>DMA Accesses</td><td>CPU Accesses</td></tr>
<tr><td>2'b11</td><td>DMA Accesses</td><td>DMA Accesses</td></tr>
</table></td></tr>
<tr><td>9
SPV
(SRAM1 Only)</td><td>Allow DMA access
0 DMA access to memory is disabled.
1 DMA access to memory is enabled.</td></tr>
<tr><td>8
WP</td><td>The Write Protect field allows only read accesses to the SRAM. When this bit is set, any attempted write access will generate an access error exception to the ColdFire processor core.
0 Allows read and write accesses to the SRAM module
1 Allows only read accesses to the SRAM module</td></tr>
<tr><td>7–6</td><td>Reserved, should be cleared.</td></tr>
<tr><td>5–1
C/I, SC, SD,
UC, UD</td><td>Address Space Masks (ASn)
These five bit fields allow certain types of accesses to be "masked," or inhibited from accessing the SRAM module. The address space mask bits are:

C/ICPU space/interrupt acknowledge cycle mask
SCSupervisor code address space mask
SDSupervisor data address space mask
UCUser code address space mask
UDUser 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 are useful for power management as detailed in Section 6.3.4, "Power Management."</td></tr>
<tr><td>0
V</td><td>The valid bit. A hardware reset clears this bit. When set, this bit enables the SRAM module; otherwise, the module is disabled.
0 Contents of RAMBAR are not valid
1 Contents of RAMBAR are valid</td></tr>
</table>

6.3.2 SRAM Initialization

After a hardware reset, the contents of the SRAM module are undefined. The valid bit of the RAMBAR is cleared, disabling the module. If the SRAM requires initialization with instructions or data, the following steps should be performed:

1. Load the RAMBAR mapping the SRAM module to the desired location within the address space and set the Valid bit.
2. Read the source data and write it to the SRAM. There are various instructions to 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 modulo-64 addresses, so this opcode generally provides maximum performance.
3. After the data has been loaded 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 emulator using the debug module can perform these initialization functions.

6.3.3 SRAM Initialization Code

The following code segment describes how to initialize the SRAM. The code sets the base address of the SRAM at $20000000 and then initializes the RAM to zeros.

```
RAMBASE EQU $20000000; set this variable to $20000000
RAMVALID EQU $00000001;
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 #4096,D0;load loop counter into D0


SRAM_INIT_LOOP:
clr.l (A0)+); clear 4 bytes of SRAM
subq.l #1,D0;decrement loop counter
bne.b SRAM_INIT_LOOP;if done, then exit; else continue looping
```

6.3.4 Power Management

As noted previously, depending on the configuration defined by the RAMBAR, instruction fetch and operand read accesses may be sent to the SRAM and unified cache simultaneously. If the access is mapped to the SRAM module, it sources the read data, and the unified cache access is discarded. If the SRAM is used only for data operands, asserting the ASn bits associated with instruction fetches can decrease power dissipation. Additionally, if the SRAM contains only instructions, masking data accesses can reduce power dissipation. Table 6-2 shows some examples of typical RAMBAR settings.

Table 6-2. Typical RAMBAR Setting Examples

Data Contained in SRAM	RAMBAR[7:0]
Code Only	$2B
Data Only	$35
Both Code And Data	$21

Chapter 7
Synchronous DRAM Controller Module

This chapter discusses the operation, memory map, register descriptions, signal and command descriptions, and an interface example for the SDRAM controller.

7.1 SDRAM Features

The key features of the SDRAM controller include the following:

- Supports one block of SDRAM
- Interface to standard synchronous dynamic random access memory (SDRAM) components
- Programmable $\overline{\text{SDRAS}}$, $\overline{\text{SDCAS}}$, and refresh timing
- Support for page mode
- Support for 16- wide SDRAM blocks

7.1.1 Block Diagram

The basic components of the DRAM controller are shown in Figure 7-1.

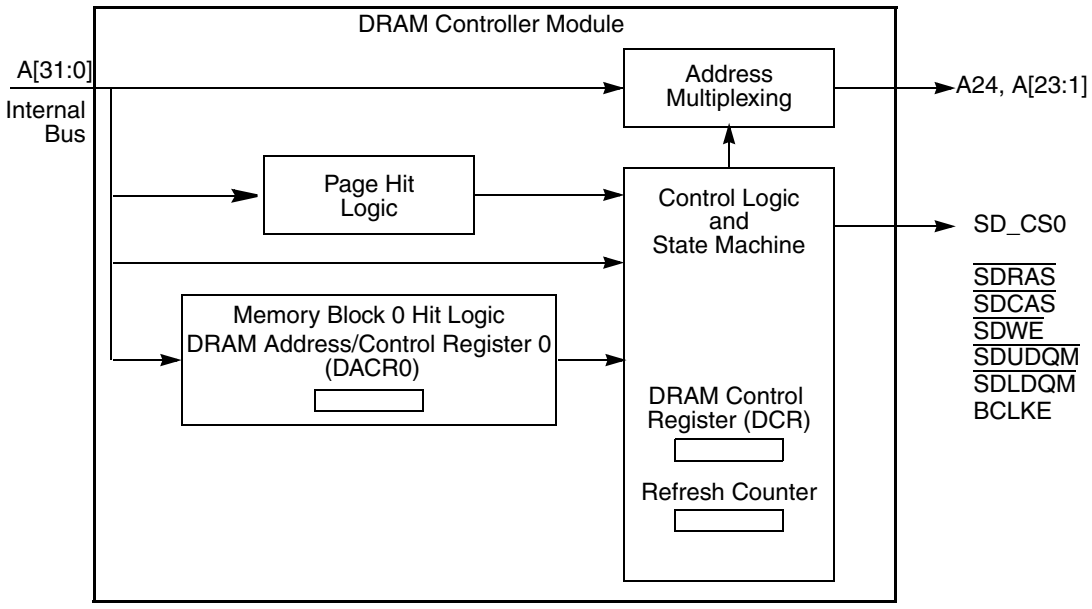


Figure 7-1. Synchronous DRAM Controller Block Diagram

The DRAM controller's major components, shown in Figure 7-1, are described as follows:

- DRAM address and control register (DACR0)—The DRAM controller consists of a configuration register unit. DACR0 is accessed at MBAR + 0x0108;
  The register information is passed on to the hit logic.
- Control logic and state machine—Generates all DRAM signals, taking bus cycle characteristic data from the block logic, along with hit information to generate DRAM accesses. Handles refresh requests from the refresh counter.
  - DRAM control register (DCR)—Contains data to control refresh operation of the DRAM controller. The memory block is refreshed concurrently as controlled by DCR[RC].
  - Refresh counter—Determines when refresh should occur, determined by the value of DCR[RC]. It generates a refresh request to the control block.
- Hit logic—Compares address and attribute signals of a current DRAM bus cycle to DACR to determine if the DRAM block is being accessed. Hits are passed to the control logic along with characteristics of the bus cycle to be generated.
- Page hit logic—Determines if the next DRAM access is in the same DRAM page as the previous one. This information is passed on to the control logic.
- Address multiplexing—Multiplexes addresses to allow column and row addresses to share pins. This allows glueless interface to DRAMs.

7.2 Synchronous Operation

By running synchronously with the system clock, the SDRAM can (after an initial latency period) be accessed on every clock; 5-1-1-1 is a typical MCF5251 burst rate to SDRAM.

NOTE

Because the MCF5251 cannot have more than one page open at a time, it does not support interleaving.

Table 7-1 lists common SDRAM commands.

Table 7-1. SDRAM Commands

Command	Definition
ACTV	Activate. Executed before READ or WRITE executes; SDRAM registers and decodes row address.
MRS	Mode register set.
NOP	No-op. Does not affect SDRAM state machine; DRAM controller control signals negated; $\overline{\text{SD\_CS0}}$ asserted.
PALL	Precharge all. Precharges all internal banks of an SDRAM component; executed before new page is opened.
READ	Read access. SDRAM registers column address and decodes that a read access is occurring.
REF	Refresh. Refreshes internal bank rows of SDRAM.
SELF	Self refresh. Refreshes internal bank rows of SDRAM when it is in low-power mode.
SELFX	Exit self refresh. This command is sent to the DRAM controller when DCR[IS] is cleared.
WRITE	Write access.

Commands are issued to memory using specific encoding on address and control pins. After system reset, a command must be sent to the SDRAM mode register to configure SDRAM operating parameters.

NOTE

Synchronous operation is selected by setting DCR[SO], DRAM controller registers reflect the synchronous operation.

7.2.1 DRAM Controller Signals in Synchronous Mode

Table 7-2 shows the behavior of DRAM signals in synchronous mode.

Table 7-2. Synchronous DRAM Signal Connections

Signal	Description
$\overline{\text{SDRAS}}$	Synchronous row address strobe. Indicates a valid SDRAM row address is present and can be latched by the SDRAM. $\overline{\text{SDRAS}}$ should be connected to the corresponding SDRAM $\overline{\text{SRAS}}$.
$\overline{\text{SDCAS}}$	Synchronous column address strobe. Indicates a valid column address is present and can be latched by the SDRAM. $\overline{\text{SDCAS}}$ should be connected to the corresponding signal labeled $\overline{\text{SCAS}}$ on the SDRAM.
$\overline{\text{SDWE}}$	DRAM read/write. Asserted for write operations and negated for read operations.
SD_CS0	Chip Select for the SDRAM memory block connected to the MCF5251.
BCLKE	Synchronous DRAM clock enable. Connected directly to the CKE (clock enable) signal of SDRAMs. Enables and disables the clock internal to SDRAM. When BCLKE is low, memory can enter a power-down mode where operations are suspended or they can enter self-refresh mode. BCLKE functionality is controlled by DCR[COC]. For designs using external multiplexing, setting COC allows BCLKE to provide command-bit functionality.
UDQM LDQM	Column address strobe. For synchronous operation, UDQM, LDQM function as byte enables to the SDRAMs. They connect to the DQM signals (or mask qualifiers) of the SDRAMs.
BCLK	Bus clock output. Connects to the CLK input of SDRAMs.

Figure 7-2 shows a typical signal configuration for synchronous mode.

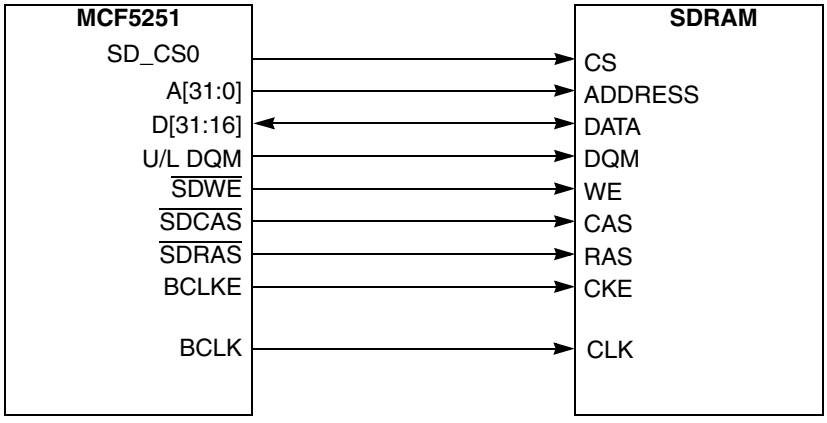


Figure 7-2. MCF5251 SDRAM Interface

7.3 SDRAM Memory Map and Register Definitions

The memory map is shown in Table 7-3. Field and bit descriptions are shown in the following sections.

7.3.1 DRAM Controller Registers

The DRAM controller registers memory map is shown in Table 7-3.

Table 7-3. DRAM Controller Registers

MBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]
0x100	DRAM control register (DCR) [Section 7.3.1, “DRAM Controller Registers”]		Reserved	
0x104	Reserved			
0x108	DRAM address and control register 0 (DACR0) [See Section 7.3.1.2, “DRAM Address and Control (DACR0) (Synchronous Mode)”]			
0x10C	DRAM mask register block 0 (DMR0) [See Section 7.3.1.3, “DRAM Controller Mask Registers (DMR0)”]			
0x110	Reserved			
0x114	Reserved			

7.3.1.1 DRAM Control Register (DCR) (Synchronous Mode)

The DRAM control register (DCR), Figure 7-3, controls refresh logic.

Address MBAR + 0x100 Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	SO		NAM	COC	IS	RTIM			RC							
Reset	0	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 7-3. DRAM Control Register (DCR) (Synchronous Mode)

Table 7-4. DRAM Control Register (DCR) Field Descriptions (Synchronous Mode)

Field	Description
15 SO	Synchronous operation. Selects synchronous or asynchronous mode. When in synchronous mode, the DRAM controller can be switched to ADRAM mode only by resetting the MCF5251. 0 Asynchronous DRAM. Default at reset. Do not use. 1 Synchronous DRAM **Note:** bit setting SO = 0 is a legacy mode. Do not use. First action must always be to set this bit.
14	Reserved, should be cleared.
13 NAM	No address multiplexing. Some implementations require external multiplexing. For example, when linear addressing is required, the DRAM should not multiplex addresses on DRAM accesses. 0 The DRAM controller multiplexes the external address bus to provide column addresses. 1 The DRAM controller does not multiplex the external address bus to provide column addresses.
12 COC	Command on SDRAM clock enable (BCLKE). Implementations that use external multiplexing (NAM = 1) must support command information to be multiplexed onto the SDRAM address bus. 0 BCLKE functions as a clock enable; self-refresh is initiated by the DRAM controller through DCR[IS]. 1 BCLKE drives command information. Because BCLKE is not a clock enable, self-refresh cannot be used (setting DCR[IS]). Thus, external logic must be used if this functionality is desired. External multiplexing is also responsible for putting the command information on the proper address bit.

Table 7-4. DRAM Control Register (DCR) Field Descriptions (Synchronous Mode) (continued)

Field	Description
11 IS	Initiate self-refresh command. 0 Take no action or issue a SELFX command to exit self refresh. 1 If DCR[COC] = 0, the DRAM controller sends a SELF command to the SDRAMv to put it in low-power, self-refresh state where they remain until IS is cleared, at which point the controller sends a SELFX command for the SDRAM to exit self-refresh. The refresh counter is suspended while the SDRAM is in self-refresh; the SDRAM controls the refresh period.
10–9 RTIM	Refresh timing. Determines the timing operation of auto-refresh in the DRAM controller. Specifically, it determines the number of clocks inserted between a REF command and the next possible ACTV command. This corresponds to t_{RC} in the SDRAM specification. 00 3 clocks 01 6 clocks 1x 9 clocks
8–0 RC	Refresh count. Controls refresh frequency. The number of bus clocks between refresh cycles is (RC + 1) * 16. Refresh can range from 16–8192 bus clocks to accommodate both standard and low-power DRAMs with bus clock operation from less than 2 MHz to greater than 50 MHz. The following example calculates RC for an auto-refresh period for 4096 rows to receive 64 mS of refresh every 15.625 µs for each row (625 bus clocks at 40 MHz). # of bus clocks = 625 = (RC field + 1) * 16 RC = (625 bus clocks/16) -1 = 38.06, which rounds to 38; therefore, RC = 0x26.

7.3.1.2 DRAM Address and Control (DACR0) (Synchronous Mode)

The DRAM address and control register (DACR0), shown in Figure 7-4, contain the base address compare value and the control bits for the memory block of the DRAM controller. Address and timing are also controlled by bits in DACR0.

Address MBAR+0x108 (DACR0) Access: User read/write

	31 30 29 28	27 26 25 24	23 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	BA						RE		CASL			CBM				IMRS		PS	IP	PM		
Reset	– – – –	– – – –	– – – –	– –	–	–	0	–	–	–	–	–	–	–	–	0	–	–	–	–	–	–

Figure 7-4. DRAM Address and Control Register (DACR0) (Synchronous Mode)

Table 7-5 describes DACR0 fields.

Table 7-5. DRAM Address and Control Register (DACR0) Field Descriptions (Synchronous Mode)

Field	Description
31–18 BA	Base address register. With DMR[BAM], determines the address range in which the associated DRAM block is located. Each BA bit is compared with the corresponding address of the current bus cycle. If all unmasked bits match, the address hits in the associated DRAM block.
17–16	Reserved, should be cleared.

Table 7-5. DRAM Address and Control Register (DACR0) Field Descriptions (Synchronous Mode) (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>15
RE</td><td>Refresh enable. Determines when the DRAM controller generates a refresh cycle to the DRAM block.
0 Do not refresh associated DRAM block
1 Refresh associated DRAM block</td></tr>
<tr><td>14</td><td>Reserved, should be cleared.</td></tr>
<tr><td>13–12
CASL</td><td>$\overline{\text{CAS}}$ latency. Affects the following SDRAM timing specifications. Timing nomenclature varies with manufacturers. Refer to the SDRAM specification for the appropriate timing nomenclature:
Note: The SDRAM controller only supports CAS latencies of 1 or 2. However very few SDRAM devices are available that support CASL = 1. So we recommend to only use CASL = 2. Some fast SDRAM are now becoming available and require a CASL = 3 which is not supported by this SDRAM controller.
<table>
<tr><th rowspan="2">Parameter</th><th colspan="4">Number of Bus Clocks</th></tr>
<tr><th>CASL = 00</th><th>CASL = 01</th><th>CASL = 10</th><th>CASL = 11</th></tr>
<tr><td>t_{RCD}—$\overline{\text{SRAS}}$ assertion to $\overline{\text{SCAS}}$ assertion</td><td>N/A</td><td>2</td><td>3</td><td>3</td></tr>
<tr><td>t_{CASL}—$\overline{\text{SCAS}}$ assertion to data out</td><td>N/A</td><td>1</td><td>2</td><td>2</td></tr>
<tr><td>t_{RAS}—ACTV command to precharge command</td><td>N/A</td><td>4</td><td>6</td><td>6</td></tr>
<tr><td>t_{RP}—Precharge command to ACTV command</td><td>N/A</td><td>2</td><td>3</td><td>3</td></tr>
<tr><td>t_{RWL},t_{RDL}—Last data input to precharge command</td><td>N/A</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>t_{EP}—Last data out to precharge command)</td><td>N/A</td><td>1</td><td>1</td><td>1</td></tr>
</table></td></tr>
<tr><td>11</td><td>Reserved, should be cleared.</td></tr>
<tr><td>10–8
CBM</td><td>Command (AP) and Bank Select MUX [2:0]. Because different SDRAM configurations cause the command and bank select lines to correspond to different addresses, these resources are programmable. CBM determines the addresses onto which these functions are multiplexed.
CBM Command Bit Bank Select Lines
000 17 18 and up
001 18 19 and up
010 19 20 and up
011 20 21 and up
100 21 22 and up
101 22 23 and up
110 23 24 and up
111 24 25 and up
This encoding and the address multiplexing scheme handle common SDRAM organizations. Bank select lines include a base line and all address lines above for SDRAMs with multiple bank select lines.</td></tr>
<tr><td>7</td><td>Reserved, should be cleared.</td></tr>
<tr><td>6
IMRS</td><td>Initiate mode register set (MRS) command. Setting IMRS generates a MRS command to the associated SDRAM. In initialization, IMRS should be set only after all DRAM controller registers are initialized and PALL and REFRESH commands have been issued. After IMRS is set, the next access to an SDRAM block programs the SDRAM's mode register. Thus, the address of the access should be programmed to place the correct mode information on the SDRAM address pins. Because the SDRAM does not register this information, it doesn't matter if the IMRS access is a read or a write. The DRAM controller clears IMRS after the MRS command finishes.
0 Take no action
1 Initiate MRS command</td></tr>
</table>

Table 7-5. DRAM Address and Control Register (DACR0) Field Descriptions (Synchronous Mode) (continued)

Field	Description
5–4 PS	Port size. Indicates the port size of the associated block of SDRAM, which allows for dynamic sizing of associated SDRAM accesses. 1x 16-bit port 0x Do not use.
3 IP	Initiate precharge all (PALL) command. The DRAM controller clears IP after the PALL command is finished. Accesses using IP should be no wider than the port size programmed in PS. 0 Take no action. 1 A PALL command is sent to the associated SDRAM block. During initialization, this command is executed after all DRAM controller registers are programmed. After IP is set, the next write to an appropriate SDRAM address generates the PALL command to the SDRAM block.
2 PM	Page mode. Indicates how the associated SDRAM block supports page-mode operation. 0 Page mode on bursts only. The DRAM controller dynamically bursts the transfer if it falls within a single page and the transfer size exceeds the port size of the SDRAM block. After the burst, the page closes and a precharge is issued. 1 Continuous page mode. The page stays open and only $\overline{\text{SDCAS}}$ needs to be asserted for sequential SDRAM accesses that hit in the same page, regardless of whether the access is a burst.
1–0	Reserved, should be cleared.

7.3.1.3 DRAM Controller Mask Registers (DMR0)

The DMR0, Figure 7-5, includes mask bits for the base address and for address attributes.

Address MBAR + 0x10C (DMR0) Access: User read/write

	31	30	29	28	27	26	25	24	23	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																							WP		C/I	AM	SC	SD	UC	UD	V
W																																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0

Figure 7-5. DRAM Controller Mask Register (DMR0)

Table 7-6. DRAM Controller Mask Register (DMR0) Field Descriptions

Bits	Description
31–18 BAM	Base address mask. Masks DACR0[BA]. Lets the DRAM controller connect to various DRAM sizes. Mask bits need not be contiguous (see Section 7.6, “SDRAM Example.”) 0 The associated address bit is used in decoding the DRAM hit to a memory block. 1 The associated address bit is not used in the DRAM hit decode.
17–9	Reserved, should be cleared.
8 WP	Write protect. Determines whether the associated block of DRAM is write protected. 0 Allow write accesses 1 Ignore write accesses. The DRAM controller ignores write accesses to the memory block and an address exception occurs. Write accesses to a write-protected DRAM region are compared in the chip select module for a hit. If no hit occurs, an external bus cycle is generated. If this external bus cycle is not acknowledged, an access exception occurs.

Table 7-6. DRAM Controller Mask Register (DMR0) Field Descriptions (continued)

<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>7</td><td>Reserved, should be cleared.</td></tr>
<tr><td>6–1
AMx</td><td>Address modifier masks. Determine which accesses can occur in a given DRAM block.
0 Allow access type to hit in DRAM
1 Do not allow access type to hit in DRAM
<table>
<tr><th>Bit</th><th>Associated Access Type</th><th>Access Definition</th></tr>
<tr><td>C/I</td><td>CPU space/interrupt acknowledge</td><td>MOVEC instruction or interrupt acknowledge cycle</td></tr>
<tr><td>AM</td><td>Alternate master</td><td>External or DMA master</td></tr>
<tr><td>SC</td><td>Supervisor code</td><td>Any supervisor-only instruction access</td></tr>
<tr><td>SD</td><td>Supervisor data</td><td>Any data fetched during the instruction access</td></tr>
<tr><td>UC</td><td>User code</td><td>Any user instruction</td></tr>
<tr><td>UD</td><td>User data</td><td>Any user data</td></tr>
</table></td></tr>
<tr><td>0
V</td><td>Valid. Cleared at reset to ensure that the DRAM block is not erroneously decoded.
0 Do not decode DRAM accesses.
1 Registers controlling the DRAM block are initialized; DRAM accesses can be decoded.</td></tr>
</table>

7.4 General Synchronous Operation Guidelines

To reduce system logic and to support a variety of SDRAM sizes, the DRAM controller provides SDRAM control signals as well as a multiplexed row address and column address to the SDRAM.

When SDRAM blocks are accessed, the DRAM controller can operate in either burst or continuous page mode. The following sections describe the DRAM controller interface to SDRAM, the supported bus transfers, and initialization.

7.4.1 Address Multiplexing

Table 7-7, Table 7-8, Table 7-9, Table 7-10, and Table 7-11 provide a comprehensive, step-by-step method to determine the correct address line connections for interfacing the MCF5251 to SDRAM. Note: that there are separate connection tables for 4Mb to 128 Mb devices and 256 Mb devices.

Specifically for the 256Mb devices the tables change due to the fact that we need to have a A24 address line. But with the MCF5251 A24 and A20 are shared on the same pin. This means that when we program the A20/A24 pin to be A24. We no longer have A20 available to any memory device connected to the memory bus.

To use the tables, find the one that corresponds to the number of column address lines on the SDRAM. Most SDRAMs likely have fewer address lines than are shown in the tables, so follow only the connections shown until all SDRAM address lines are connected.

It is important to ensure that the Command (CBM) or AP control signal is connected to correct Address line on the selected SDRAM. The address line used (or position) for the AP control signal is programmable via bits 10:8 of the DACRx register.

In the case of a Samsung SDRAM (K4S641633D-G) which has 8 column address lines the AP address pin is A10 (the address line used for the CBM/AP control signal may differ on other manufacturer devices). We choose to use A19 for the CBM/AP control signal and connect it to A10 on the SDRAM. We then set DACRx bits 10:8 to 010b which selects A19 as the CBM/AP mapping the Bank Select pins above this.

Note: for SDRAM's with more column address lines the position of the CBM / AP control signal moves up one address line in each case.

The following SDRAM pin connection tables are for use with 4Mbit, 16Mbit, 64Mbit or 128Mbit devices only.

Table 7-7. SDRAM Interface (16-Bit Port, 8-Column Address Lines)

MCF5251 Pins	A16	A15	A14	A13	A12	A11	A10	A9	A17	A18	A19	A20	A21	A22	A23
Row	16	15	14	13	12	11	10	9	17	18	19	20	21	22	23
Column	1	2	3	4	5	6	7	8	–	–	–	–	–	–	–
SDRAM Pins	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14

Table 7-8. SDRAM Interface (16-Bit Port, 9-Column Address Lines)

MCF5251 Pins	A16	A15	A14	A13	A12	A11	A10	A9	A18	A19	A20	A21	A22	A23
Row	16	15	14	13	12	11	10	9	18	19	20	21	22	23
Column	1	2	3	4	5	6	7	8	17	–	–	–	–	–
SDRAM Pins	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13

Table 7-9. SDRAM Interface (16-Bit Port, 10-Column Address Lines

MCF5251 Pins	A16	A15	A14	A13	A12	A11	A10	A9	A18	A20	A21	A22	A23
Row	16	15	14	13	12	11	10	9	18	20	21	22	23
Column	1	2	3	4	5	6	7	8	17	19	–	–	–
SDRAM Pins	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12

The following SDRAM pin connection tables are for use with 256 Mbit devices only.

For the 256 Mbyte SDRAM, there is the restriction that address line A20 cannot be output. (we need A24). If you want consistent mapping of the D-RAM in ColdFire memory space, A20 must be sent to the D-RAM. To get this done, the only way is to make sure the D-RAM controller outputs A20 during the CAS phase on A21, and ensure address pin A21 is connected to a D-RAM pin that uses both row and column address. In this case, there is only one such pin, and this is A8 (CAS8/RAS8). Also, to get the column address on A21, the AP bit must be set above this, lowest we can set it is A22. So, AP must be connected to A22. Bank addresses are then A23 and above. (Bank addresses need equal address during CAS and RAS phase). The remaining 3 D-RAM ras-only address lines are connected to A9, A11 and A12.

Table 7-10. SDRAM Interface (16-Bit Port, 9-Column Address Lines)

MCF5251 Pins	**A16**	**A15**	**A14**	**A13**	**A12**	**A11**	**A10**	**A9**	**A21**	**A17**	**A18**	**A19**	**A22**	**A23**	**A24**
Row	16	15	14	13	12	11	10	9	21	17	18	19	22	23	24
Column	1	2	3	4	5	6	7	8	20	–	–	–	–	–	–
Output during CAS	A1	A2	A3	A14	A5	A6	A7	A8	A20	A16	A17	A18	AP	A23	A24
Output during RAS	A16	A15	A14	A13	A12	A11	A10	A9	A21	A17	A18	A19	A22	A23	A24
SDRAM Pins	**A0**	**A1**	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	**A8**	**A9**	**A11**	**A12**	**A10**	**A13**	**A14**

7.4.2 Interfacing Example

The tables in the previous section can be used to configure the interface in the following example. To interface one 1M × 16-bit × 4 bank SDRAM component (8 columns) to the MCF5251, the connections would be as shown in Table 7-11. Pin A20/A24 is programmed to A20 mode and A19 is programmed as the AP / CMD.

Table 7-11. SDRAM Hardware Connections

SDRAM Pins	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10=CMD	A11	BA0/A12	BA1/A13
MCF5251 Pins	A16	A15	A14	A13	A12	A11	A10	A9	A17	A18	A19	A20/A24	A21	A22

7.4.3 Burst Page Mode

The SDRAM can efficiently provide data when an SDRAM page is opened. As soon as $\overline{\text{SDCAS}}$ is issued, the SDRAM accepts a new address and asserts $\overline{\text{SDCAS}}$ every clock for as long as accesses are in that page. In burst page mode, there are multiple read or write operations for every ACTV command in the SDRAM if the requested transfer size exceeds the port size of the associated SDRAM. The first cycle of the transfer generates the ACTV and READ or WRITE commands; additional cycles generate only READ or WRITE commands. As soon as the transfer completes, the PALL command is generated to prepare for the next access.

NOTE

In synchronous operation, burst mode and address incrementing during burst cycles are controlled by the MCF5251 DRAM controller. Thus, instead of the SDRAM enabling its internal burst incrementing capability, the MCF5251 controls this function. This means that the burst function that is enabled in the mode register of SDRAMs must be disabled when interfacing to the MCF5251.

Figure 7-6 shows a burst read operation. In this example, DACR[CASL] = 01, for an $\overline{\text{SRAS}}$-to-$\overline{\text{SCAS}}$ delay (t_{RCD}) of 1 BCLKO cycle. Because t_{RCD} is one more than the read CAS latency ($\overline{\text{SCAS}}$ assertion to data out), this value is 2 BCLK cycles. Notice that NOPs are executed until the last data is read. A PALL command is executed one cycle after the last data transfer.

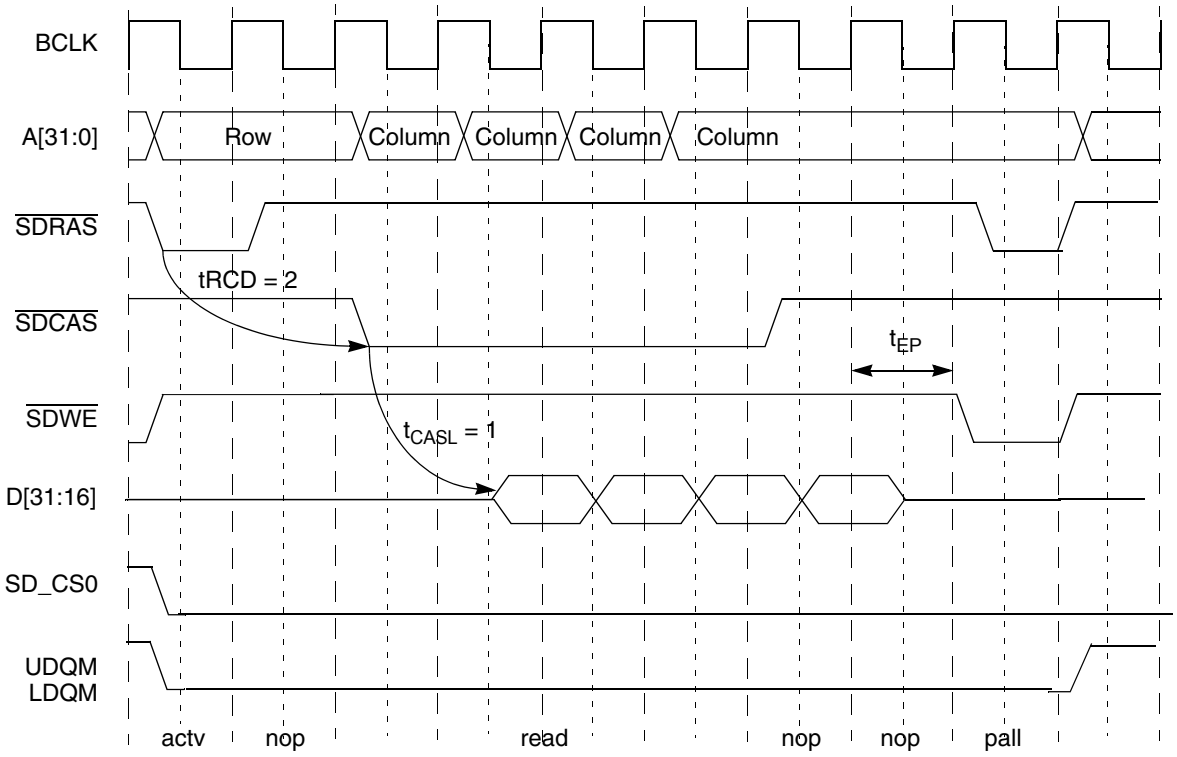


Figure 7-6. Burst Read SDRAM Access

Figure 7-7 shows the burst write operation. In this example, DACR[CASL] = 01, which creates an $\overline{\text{SRAS}}$-to-$\overline{\text{SCAS}}$ delay (t_{RCD}) of 2 BCLK cycles.

NOTE

Data is available upon $\overline{\text{SCAS}}$ assertion and a burst write cycle completes two cycles sooner than a burst read cycle with the same t_{RCD}. The next bus cycle is initiated sooner, but cannot begin an SDRAM cycle until the precharge-to-ACTV delay completes.

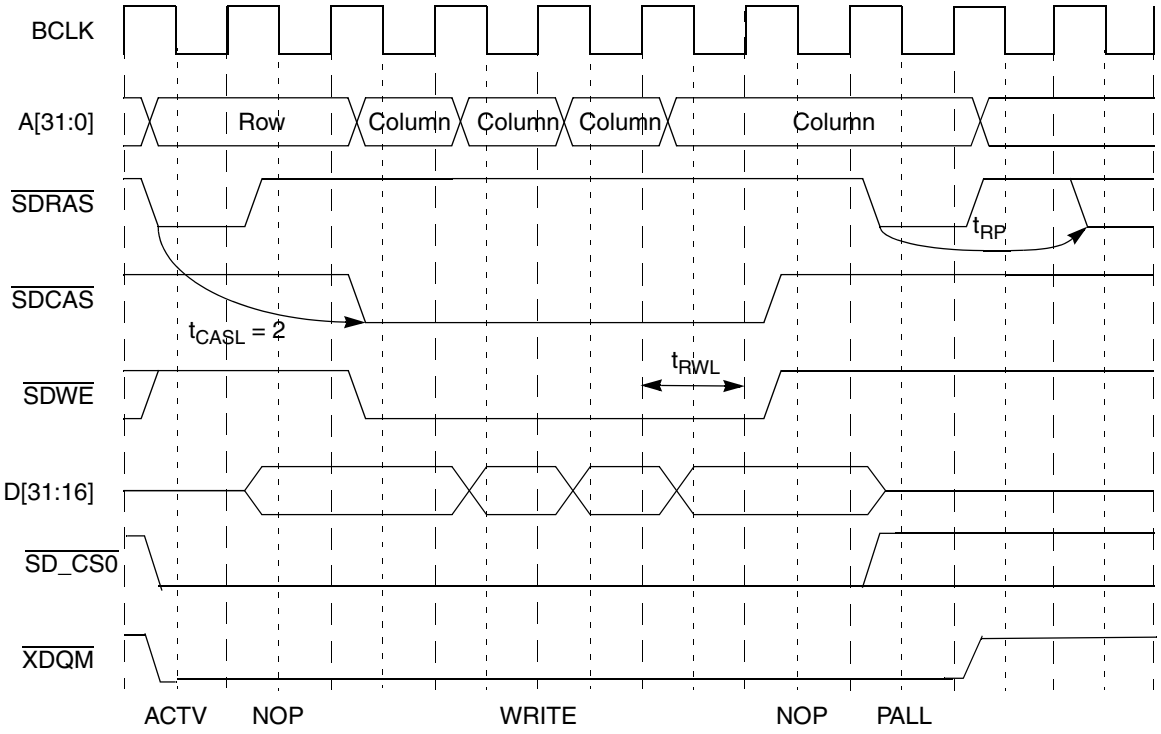


Figure 7-7. Burst Write SDRAM Access

Accesses in synchronous burst page mode always cause the following sequence:

1. ACTV command
2. NOP commands to assure $\overline{\text{SRAS}}$-to-$\overline{\text{SCAS}}$ delay (if $\overline{\text{CAS}}$ latency is 1, there are no NOP commands)
3. Required number of READ or WRITE commands to service the transfer size with the given port size
4. Some transfers need more NOP commands to assure the ACTV-to-precharge delay
5. PALL command
6. Required number of idle clocks inserted to assure precharge-to-ACTV delay

7.4.4 Continuous Page Mode

Continuous page mode is identical to burst page mode, except that it allows the processor core to handle successive bus cycles that hit the same page without having to close the page. When the current bus cycle finishes, the MCF5251 core internal pipelined bus can predict whether the upcoming cycle will hit in the same page.

- If the next bus cycle is not pending or misses in the page, the PALL command is generated to the SDRAM.
- If the next bus cycle is pending and hits in the page, the page is left open, and the next SDRAM access begins with a READ or WRITE command.
- Because of the nature of the internal CPU pipeline this condition does not occur often; however, the use of continuous page mode is recommended because it can provide a slight performance increase.

Figure 7-8 shows two read accesses in continuous page mode.

NOTE

There is no precharge between the two accesses. Also, the second cycle begins with a read operation with no ACTV command.

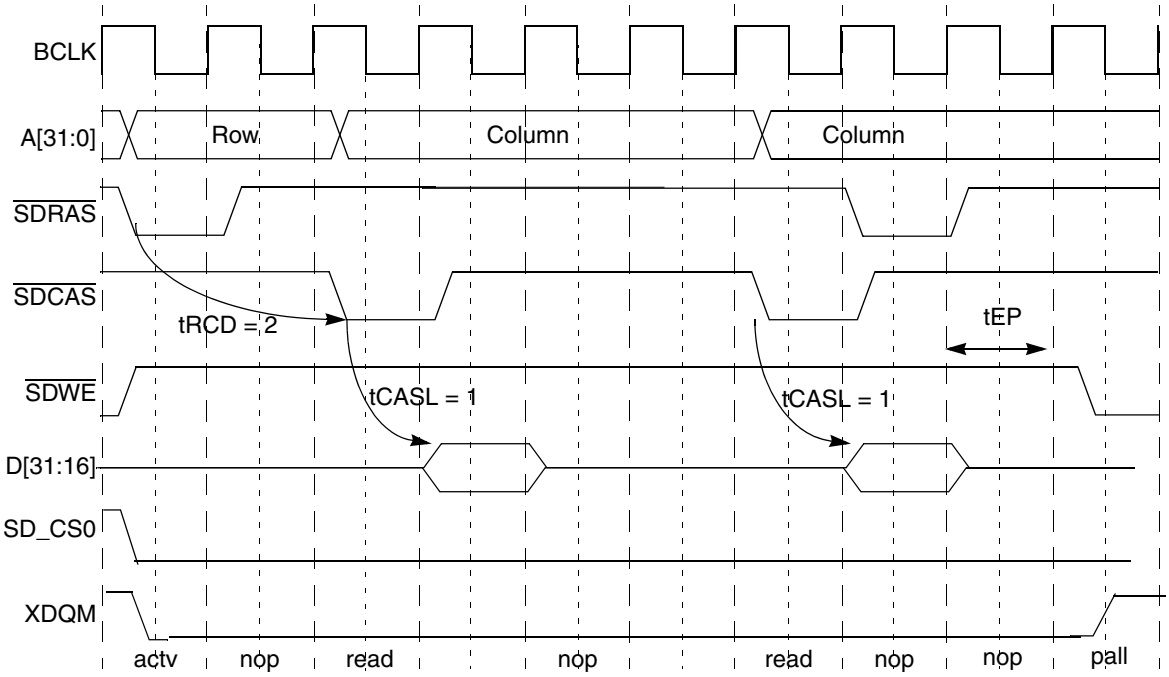


Figure 7-8. Synchronous, Continuous Page-Mode Access—Consecutive Reads

Figure 7-9 shows a write followed by a read in continuous page mode. Because the bus cycle is terminated with a WRITE command, the second cycle begins sooner after the write than after the read. A read requires data to be returned before the bus cycle can terminate.

NOTE

In continuous page mode, secondary accesses output the column address only.

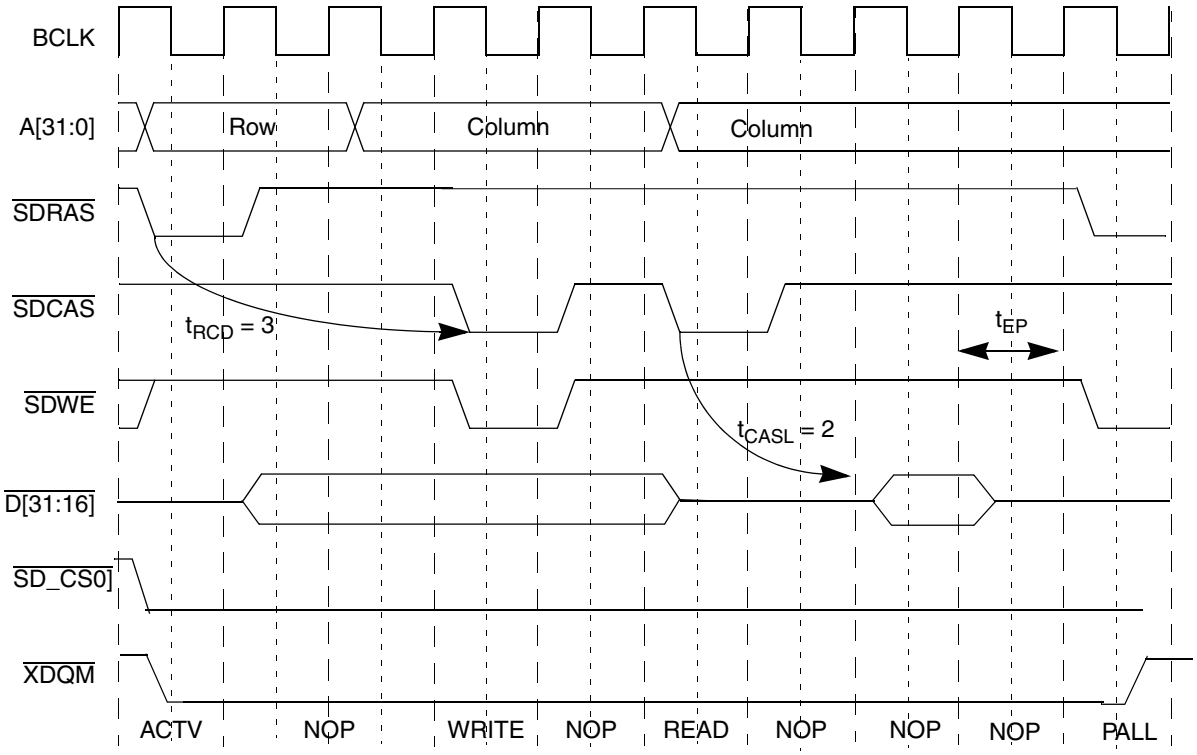


Figure 7-9. Synchronous, Continuous Page-Mode Access—Read after Write

7.4.5 Auto-Refresh Operation

The DRAM controller is equipped with a refresh counter and control. This logic is responsible for providing timing and control to refresh the SDRAM. Once the refresh counter is set, and refresh is enabled, the counter counts to zero. At this time, an internal refresh request flag is set and the counter begins counting down again. The DRAM controller completes any active burst operation and then performs a PALL operation. The DRAM controller then initiates a refresh cycle and clears the refresh request flag. This refresh cycle includes a delay from any precharge to the auto-refresh command, the auto-refresh command, and then a delay until any ACTV command is allowed. Any SDRAM access initiated during the auto-refresh cycle is delayed until the cycle is completed.

Figure 7-10 shows the auto-refresh timing. In this case, there is an SDRAM access when the refresh request becomes active. The request is delayed by the precharge to ACTV delay programmed into the active SDRAM bank by the CAS bits. The REF command is then generated and the delay required by DCR[RTIM] is inserted before the next ACTV command is generated. In this example, the next bus cycle is initiated, but does not generate an SDRAM access until T_{RC} is finished.

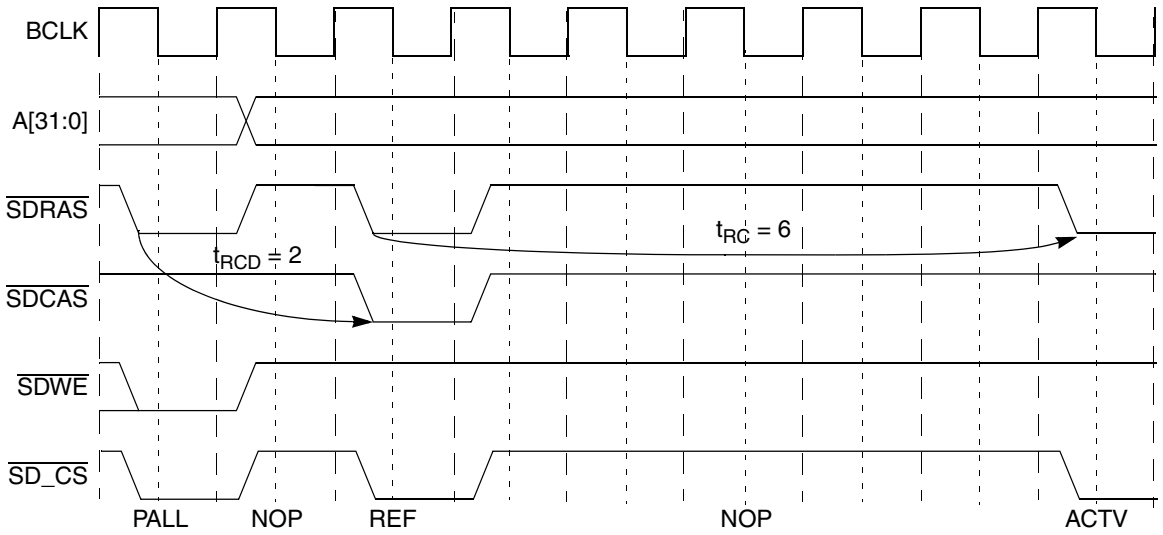


Figure 7-10. Auto-Refresh Operation

7.4.6 Self-Refresh Operation

Self-refresh is a method of allowing the SDRAM to enter into a low-power state, while at the same time to perform an internal refresh operation and to maintain the integrity of the data stored in the SDRAM. The DRAM controller supports self-refresh with DCR[IS]. When IS is set, the SELF command is sent to the SDRAM. When IS is cleared, the SELFX command is sent to the DRAM controller. Figure 7-11 shows the self-refresh operation.

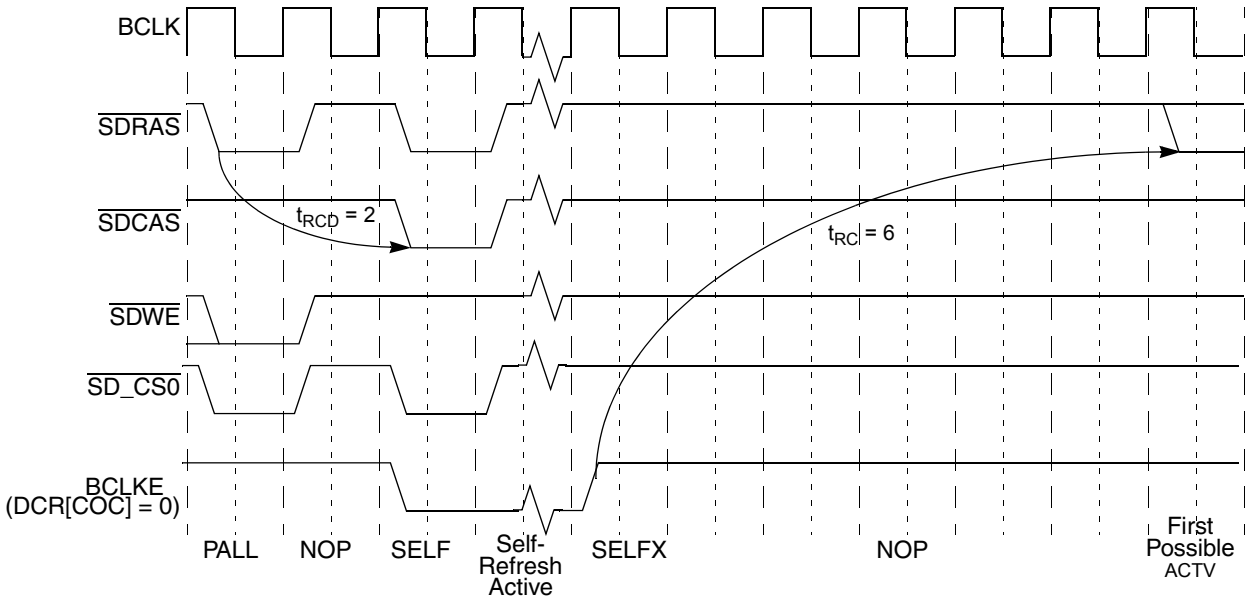


Figure 7-11. Self-Refresh Operation

7.5 Initialization Sequence

Synchronous DRAMs have a prescribed initialization sequence. The DRAM controller supports this sequence with the following procedure:

1. SDRAM control signals are reset to idle state. Wait the prescribed period after reset before any action is taken on the SDRAMs. This is normally around 100 µs.
2. Initialize the DCR, DACR, and DMR in their operational configuration. Do not yet enable PALL or REF commands.
3. Issue a PALL command to the SDRAMs by setting DCR[IP] and accessing a SDRAM location. Wait the time (determined by t_{RP}) before any other execution.
4. Enable refresh (set DACR[RE]) and wait for at least 8 refreshes to occur.
5. Before issuing the MRS command, determine if the DMR mask bits need to be modified to allow the MRS to execute properly.
6. Issue the MRS command by setting DACR[IMRS] and accessing a location in the SDRAM.

NOTE

Mode register settings are driven on the SDRAM address bus, so care must be taken to change DMR[BAM] if the mode register configuration does not fall in the address range determined by the address mask bits. After the mode register is set, DMR mask bits can be restored to their desired configuration.

7.5.1 Mode Register Settings

It is possible to configure the operation of SDRAMs, namely their burst operation and $\overline{\text{CAS}}$ latency, through the SDRAM mode register. $\overline{\text{CAS}}$ latency is a function of the speed of the SDRAM and the bus clock of the DRAM controller. The DRAM controller operates at a $\overline{\text{CAS}}$ latency of 1 or 2.

Although the MCF5251 DRAM controller supports bursting operations, it does not use the bursting features of the SDRAMs. Because the MCF5251 can burst operand sizes of 1, 2, 4, or 16 bytes long, the concept of a fixed burst length in the SDRAMs mode register becomes problematic. Therefore, the MCF5251 DRAM controller generates the burst cycles rather than the SDRAM device. Because the MCF5251 generates a new address and a READ or WRITE command for each transfer within the burst, the SDRAM mode register should be set either to a burst length of one or to not burst. This allows bursting to be controlled by the MCF5251 instead.

The SDRAM mode register is written to by setting the associated block's DACR[IMRS]. First, the base address and mask registers must be set to the appropriate configuration to allow the mode register to be set.

NOTE

Improperly set DMR mask bits may prevent access to the mode register address. Thus, the user should determine the mapping of the mode register address to the MCF5251 address bits to find out if an access is blocked. If the DMR setting prohibits mode register access, the DMR should be reconfigured to enable the access and then set to its necessary configuration after the MRS command executes.

The associated CBM bits should also be initialized. After DACR[IMRS] is set, the next access to the SDRAM address space generates the MRS command to that SDRAM. The address of the access should be selected to place the correct mode information on the SDRAM address pins. The address is not multiplexed for the MRS command. The MRS access can be a read or write. The important thing is that the address output of that access needs the correct mode programming information on the correct address bits.

Figure 7-12 shows the MRS command, which occurs in the first clock of the bus cycle.

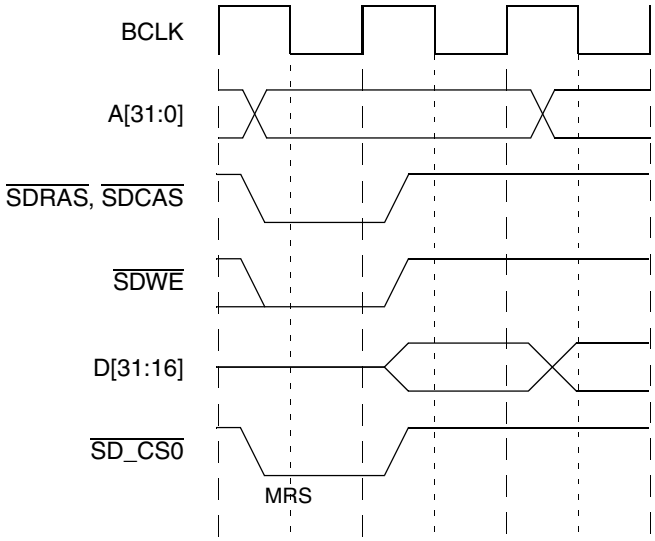


Figure 7-12. Mode Register Set (MRS) Command

7.6 SDRAM Example

This example interfaces a Samsung K4S641633 1M x 16-bit x 4 bank SDRAM component to a MCF5251 operating at 80 MHz (40 MHz bus). Table 7-12 lists design specifications for this example.

Table 7-12. SDRAM Example Specifications

Parameter	Specification
12 rows, 8 columns	–
Two bank-select lines to access four internal banks	–
ACTV-to-read/write delay (t_{RCD})	20 nS (min.)
Period between auto refresh and ACTV command (t_{RC})	70 nS
ACTV command to precharge command (t_{RAS})	48 nS (min.)
Precharge command to ACTV command (t_{RP})	20 nS (min.)
Last data input to PALL command (t_{RWL})	1 bus clock (25 nS)
Auto refresh period for 4096 rows (t_{REF})	64 mS

7.6.1 SDRAM Interface Configuration

To interface this component to the MCF5251 DRAM controller, use the connection table that corresponds to a 16-bit port size with 8 columns (Figure 7-14). Two pins select one of four banks when the part is functional. Table 7-13 shows the proper hardware hook-up.

Table 7-13. SDRAM Hardware Connections

MCF5251 Pins	A16	A15	A14	A13	A12	A11	A10	A9	A17	A18	A19	A20/A24	A21	A22
SDRAM Pins	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10 = CMD	A11	BA0	BA1

7.6.2 DCR Initialization

At power-up, the DCR has the following configuration if synchronous operation and SDRAM address multiplexing is desired.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	SO		NAM	COC	IS	RTIM			RC							
Setting	1	–	0	0	0	0	0	0	0	0	0	1	0	0	1	0
(hex)	8				0				1				2			

Figure 7-13. Initialization Values for DCR

This configuration results in a value of 0x8012 for DCR, as shown in Table 7-14.

Table 7-14. DCR Initialization Values

Bits	Setting	Description
15 SO	1	Indicating synchronous operation
14	x	Don't care (reserved)
13 NAM	0	Indicating SDRAM controller multiplexes address lines internally
12 COC	0	BCLKE is used as clock enable instead of command bit because user is not multiplexing address lines externally and requires external command feed.
11 IS	0	At power-up, allowing power self-refresh state is not appropriate because registers are being set up.
10–9 RTIM	00	Because t_{RC} value is 70 nS, indicating a 3-clock refresh-to-ACTV timing.
8–0 RC	0x12	Specification indicates auto-refresh period for 4096 rows to be 64 mS or refresh every 15.625 µs for each row, or 312 bus clocks at 40MHz. Because DCR[RC] is incremented by 1 and multiplied by 16, RC = (312 bus clocks/16) -1 = 18.56 = 0x12

7.6.3 DACR Initialization

As shown in Figure 7-14, in this example the SDRAM is programmed to access only the second 512 Kbyte block of each 1 Mbyte partition in the SDRAM (each 16 Mbyte). The starting address of the SDRAM is 0xFF80_0000. Continuous page mode feature is used.

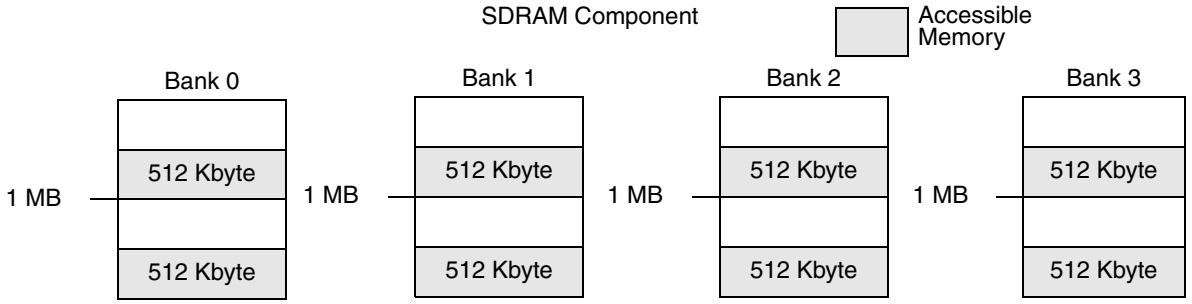


Figure 7-14. SDRAM Configuration

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Field	BA															
Setting	1111_1111_1000_10														–	–
(hex)	F				F				8				8			

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	RE		CASL			CBM				IMRS	PS		IP	PM		
Setting	0	–	01		–	010			–	0	10		0	1	–	–
(hex)	1				2				2				4			

Figure 7-15. DACR Register Configuration

This configuration results in a value of DACR0 = 0xFF88_1224, as described in Table 7-15.

Table 7-15. DACR Initialization Values

Field	Setting	Description
31–18 BA	–	Base address. So DACR0[31–16] = 0xFF88, which places the starting address of the SDRAM accessible memory at 0xFF88_0000.
17–16	–	Reserved. Don't care.
15 RE	0	0, which keeps auto-refresh disabled because registers are being set up at this time.
14	–	Reserved. Don't care.
13–12 CASL	01	Indicates a delay of data 1 cycle after $\overline{\text{CAS}}$ is asserted.
11	–	Reserved. Don't care.
10–8 CBM	010	Command bit is pin 19 and bank selects are 20 and up.

Table 7-15. DACR Initialization Values (continued)

Field	Setting	Description
7	–	Reserved. Don't care.
6 IMRS	0	Indicates MRS command has not been initiated.
5–4 PS	10	16-bit port.
3 IP	0	Indicates precharge has not been initiated.
2 PM	1	Indicates continuous page mode.
1–0	–	Reserved. Don't care.

7.6.4 DMR Initialization

In this example, only the second 512 Kbyte block of each 1 Mbyte space is accessed in each bank. In addition the SDRAM component is mapped only to readable and writable supervisor and user data. The DMRs have the following configuration.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Field	BAM															
Setting	0	0	0	0	0	0	0	0	0	1	1	1	0	1	–	–
(hex)	0				0				7				4			

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field								WP		C/I	AM	SC	SD	UC	UD	V
Setting	–	–	–	–	–	–	–	0	–	1	1	1	0	1	0	1
(hex)	0				0				7				5			

Figure 7-16. DMR0 Register

With this configuration, the DMR0 = 0x0074_0075, as described in Table 7-16.

Table 7-16. DMR0 Initialization Values

Bits	Setting	Description
31–16 BAM	–	With bits 17 and 16 as don't cares, BAM = 0x0074, which leaves bank select bits and upper 512K select bits unmasked. Bits 22 and 21 are set because they are used as bank selects; bit 20 is set because it controls the 1-MB boundary address.
15–9	–	Reserved. Don't care.
8 WP	0	Allow reads and writes
7	–	Reserved

Table 7-16. DMR0 Initialization Values (continued)

Bits	Setting	Description
6 C/I	1	Disable CPU space access
5 AM	1	Disable alternate master access
4 SC	1	Disable supervisor code accesses
3 SD	0	Enable supervisor data accesses
2 UC	1	Disable user code accesses
1 UD	0	Enable user data accesses
0 V	1	Enable accesses

7.6.5 Mode Register Initialization

When DACR[IMRS] is set, a bus cycle initializes the mode register. If the mode register setting is read on A[9:0] of the SDRAM on the first bus cycle, the bit settings on the corresponding MCF5251 address pins must be determined while being aware of masking requirements.

Table 7-17 lists the desired initialization setting:

Table 7-17. Mode Register Initialization

MCF5251 Pins	SDRAM Pins	Mode Register Initialization	
A22	BA1 / A13	–	0
A21	BA0 / A12	–	0
A20	A11	Reserved	0
A19	command / A10	WB	0
A18	A9	Opmode	0
A17	A8	Opmode	0
A9	A7	–	–
A10	A6	CASL	0
A11	A5	CASL	0
A12	A4	CASL	1
A13	A3	BT	0
A14	A2	BL	0

Table 7-17. Mode Register Initialization (continued)

MCF5251 Pins	SDRAM Pins	Mode Register Initialization	
A22	BA1 / A13	–	0
A21	BA0 / A12	–	0
A20	A11	Reserved	0
A15	A1	BL	0
A16	A0	BL	0

Next, this information is mapped to an address to determine the hexadecimal value.

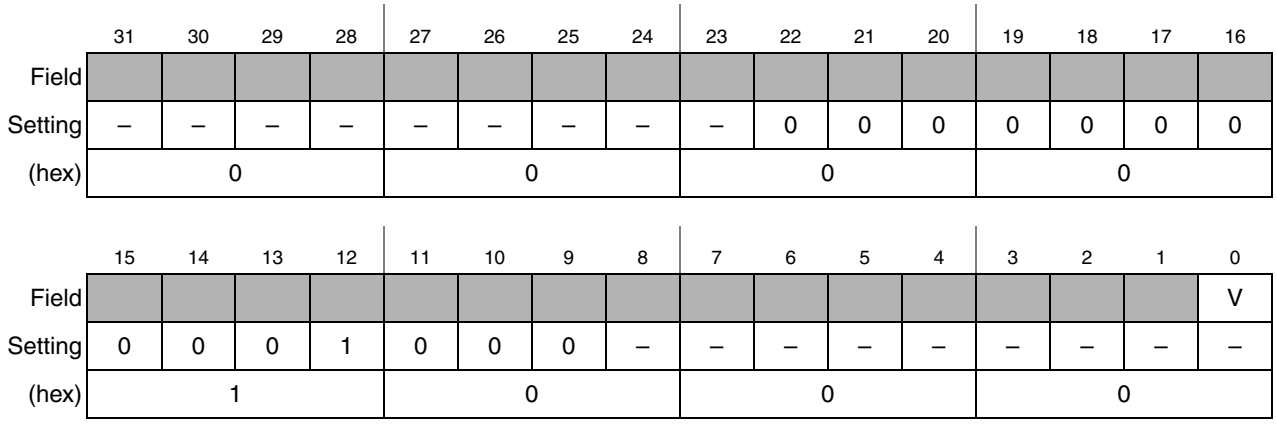


Figure 7-17. Mode Register Mapping to MCF5251 A[31:0]

Although A[31:20] corresponds to the address programmed in DACR0, according to how DACR0 and DMR0 are initialized, bit 19 must be set to hit in the SDRAM. Thus, before the mode register bit is set, DMR0[19] must be set to enable masking.

7.6.6 Initialization Code

The following assembly code initializes the SDRAM example.

Power-Up Sequence:

```
move.w   #0x8012, d0                                    //Initialize DCR
move.w   d0, DCR
move.l   #0xFF881220, d0                               //Initialize DACR0
move.l   d0, DACR0
move.l   #0x00740075, d0                                //Initialize DMR0
move.l   d0, DMR0
```

Precharge Sequence:

```
move.l   #0xFF881228, d0                                  //Set DACR0[IP]
move.l   d0, DACR0
move.l   #0xBEADDEED, d0          //Write to memory location to init. precharge
move.l   d0, 0xFF880000
```

Refresh Sequence:

```
move.l   #0xFF889220, d0                          //Enable refresh bit in DACR0
move.l   d0, DACR0
```

Mode Register Initialization Sequence:

```
move.l   #0x00600075, d0                                  //Mask bit 19 of address
move.l   d0, DMR0
move.l   #0xFF889260, d0              //Enable DACR0[IMRS]; DACR0[RE] remains set
move.l   d0, DACR0
move.l   #0x00000000, d0       //Access SDRAM address to initialize mode register
move.l   d0, 0xFF801000
move.l   #0x00740075, d0                                        //Set up DMR again
move.l   d0, DMR0
```

Chapter 8
Bus Operation

This chapter describes bus functionality, the bus control signals, and the bus cycles provided for data-transfer operations. Bus operation is defined for transfers initiated by the MCF5251 as a bus master and for transfers initiated by an alternate bus master. This chapter includes descriptions of the error conditions, bus arbitration, and the reset operation.

8.1 Bus Features

The MCF5251 bus operates using the following features:

- 24 bit address bus (24-bit for SDRAM access only 23-bit otherwise)
- 16 bit data bus
- 16 bit port size
- Generates byte, word, longword, and line size transfers
- Burst and burst-inhibited transfer support
- Internal termination generation (TA)

8.2 Bus and Control Signals

Although the timing of these signals is referenced to the BCLK, it is not considered a bus signal. It is expected that the clock will be routed as needed to meet application requirements. Table 8-1 summarizes the MCF5251 bus signals. A brief description of the function of each signal follows.

NOTE

An overbar indicates an active-low signal.

Table 8-1. MCF5251 Bus Signal Summary

Signal Name	Direction	Description
A[24:1]	Out	Address bus
R$\overline{W}$	Out	Read-write control
D[31:16]	In/Out	Data bus
$\overline{CS0}$/$\overline{CS4}$	Out	Chip select 0 / Chip select 4
$\overline{CS1}$/QSPI_CS3/GPIO28	Out	Chip select 1
$\overline{OE}$	Out	Output enable

8.2.1 Address Bus

The address bus A[24:1] provides the address of the byte or most significant byte of the word or longword being transferred.The address lines also serve as the SDRAM address pins, providing multiplexed row and column address signals.

NOTE

For SDRAM access A24 is multiplexed with A20.

A0 is not available on the address bus. As a result, the MCF5251 supports only 16-bit port size.

8.2.2 Read/Write Control

The read/write ($R\overline{W}$) control line will indicate that a bus cycle in progress is read or write. $R\overline{W}$ timing is same as address timing.

8.2.3 Transfer Acknowledge ($\overline{TA}$)

This active-low synchronous input signal indicates the successful completion of a requested data transfer operation. During MCF5251-initiated transfers, transfer acknowledge ($\overline{TA}$) is an asynchronous input signal from the referenced slave device indicating completion of the transfer.

The MCF5251 edge-detects and re-times the $\overline{TA}$ input. This means that an additional wait state may or may not be inserted. For example, if the active chip select is used to immediately generate the $\overline{TA}$ input, one or two wait states may be inserted in the bus access.

The $\overline{TA}$ signal function is not available after reset. It must be enabled by configuring the appropriate pin configuration register bit (it is multiplexed with GPIO12) along with the value of CSCRn[WS]. If $\overline{TA}$ is not used, it should either have a pull-up resistor or be driven through gating logic that always ensures the input is inactive. $\overline{TA}$ should be negated on the negating edge of the active chip select.

$\overline{TA}$ must always be negated before it can be recognized as asserted again. If held asserted into the following bus cycle, it has no effect and does not terminate the bus cycle.

$\overline{TA}$ is not used for termination during SDRAM accesses.

8.2.4 Data Bus

The data bus D[31:16] is a bidirectional, non-multiplexed bus. Data is latched by the MCF5251 on the rising BCLK clock edge. When interfacing with external memory or peripherals, the data bus port width, wait states, and internal termination are initially defined.

Table 8-2. Reset Port Settings

Reset Port Size	16 Bit
Reset cycle length	Internal termination, 15 wait cycles

The port width for each chip-select and DRAM bank are user programmable. If none of the chip-selects, DRAM bank or System Bus Controller (SBC) spaces match the address decode, the memory cycle will

terminate with error. The data bus can transfer byte or word-sized data. All 16 bits of the data bus are driven during writes, regardless of port width or operand size.

8.2.5 Chip Selects

Chip select $\overline{CS1}$ is shared with QSPI_CS3 and GPIO28.
Power-on reset function of $\overline{CS1}$/QSPI_CS3/GPIO28 is $\overline{CS1}$
The function can be programmed in the Pin Configuration register.

Chip select $\overline{CS0}$ shares with $\overline{CS4}$.
Its default mode is dependent on the state of address pin A23 at power-on reset.

This is determined as follows:

During power-on reset, logic level of pins A23 and A20/A24 are sensed. A pull-up / pull-down resistor should be connected between these pins and VDD or GND. Depending whether a pull-up or pull-down is mounted, different options are selected.

Table 8-3. Chip Select Settings

Pin	Description
A23	Pull-up: Boot from memory connected to CS0/CS4. CS0/CS4 function is CS0 Pull-down: Boot from on-chip boot ROM. CS0/CS4 function becomes CS4

When the address decode matches one of the chip select spaces, the MCF5251 processor will pull low the appropriate chip select low indicating an external bus access.

$\overline{CS2}$ is also available but is associated with the IDE read and write strobes $\overline{IDE\_DIOR}$ and $\overline{IDE\_DIOW}$.

Configuration registers for CS3 are present but no hardware pin exists for this CS on the MCF5251. However it is possible to program $\overline{BUFENB2}$ via the CS3 registers.

8.2.6 Output Enable

The $\overline{OE}$ pin on the MCF5251 will be pulled low during any read cycle from a device selected by $\overline{CS0}$, $\overline{CS1}$, $\overline{CS2}$, or $\overline{CS4}$.

8.3 Clock and Reset Signals

These signals provide the external system interface for the MCF5251 (see Table 8-4).

Table 8-4. CF-Bus Signal Summary

Signal Name	Direction	Description
RSTI	In	Reset In
BCLK	Out	System Bus Clock Output (SYSCLK)

8.3.1 Reset In

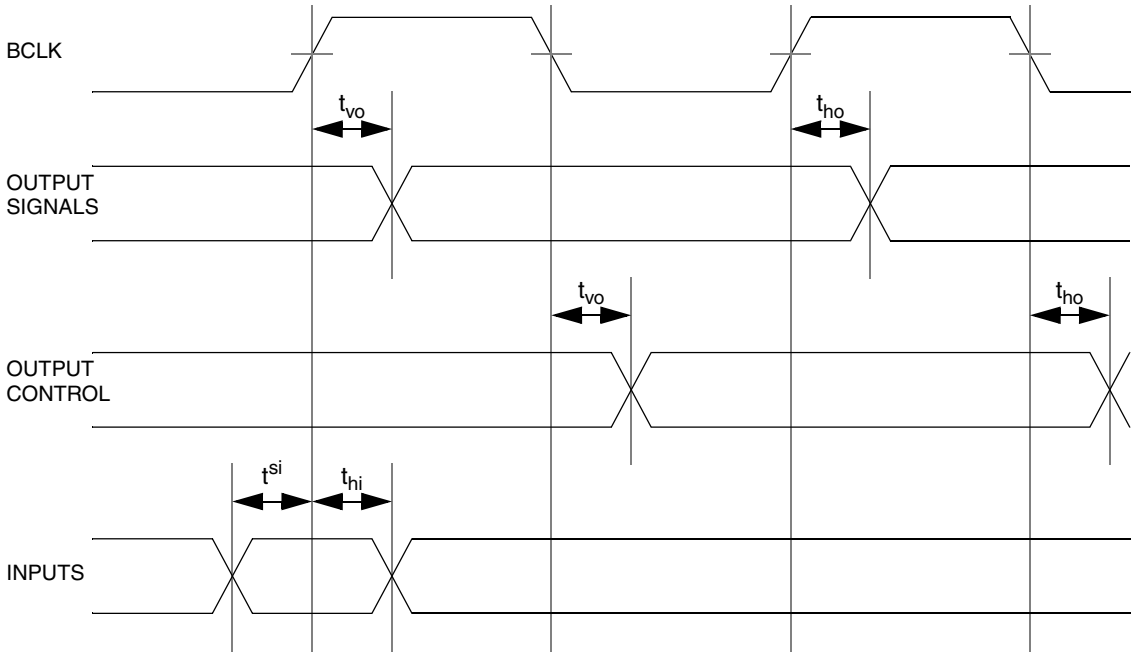
Asserting $\overline{\text{RSTI}}$ causes the MCF5251 processor to enter reset exception processing. When $\overline{\text{RSTI}}$ is recognized, the data bus is tri-stated, and $\overline{\text{OE}}$, $\overline{\text{CS0}}$, and $\overline{\text{CS1}}$ are negated. See Section 8.7, "Reset Operation."

8.3.2 System Bus Clock Output

The BCLK output signal is generated by the internal PLL, and is the system bus clock output used as the bus timing reference by the external devices. BCLK is always half the frequency of the processor clock.

8.4 Bus Characteristics

The external bus operates at the same speed as the bus clock rate, where all bus operations are synchronous to the rising edge of BCLK, and the bus chip selects are synchronous to the falling edge of the BCLK, which is shown in Figure 8-1. The bus characteristics may be somewhat different for interfacing with external DRAM.



t_{vo} = Propagation delay of signal relative to BCLK edge
t_{ho} = Output hold time relative to BCLK edge
t_{si} = Required input setup time relative to BCLK edge
t_{hi} = Required input hold time relative to BCLK edge

Figure 8-1. Signal Relationship to BCLK for Non-DRAM Access

8.5 Data Transfer Operation

Data transfer between the MCF5251 processor and other devices involves the following signals:

1. Address bus (A[23:1])
2. $\overline{\text{RW}}$ control

3. Data bus (D[31:16])
4. Strobe ($\overline{CSx}$, $\overline{OE}$)

The bus signals transition on the rising edge of BCLK. The strobe signals $\overline{CSx}$, $\overline{OE}$ make their transitions on the falling edge of BCLK. Read data is latched on the rising edge of BCLK.

The bus supports byte, word, and longword operand transfers and uses a 16-bit data port. With the MCF5251, the port size of all memory must be programmed to 16 bits, the internal transfer termination must be enabled, and the number of wait states must be set for the external slave being accessed by programming the Chip-Select Control Registers (CSCRs) and the DRAM Controller Control Registers (DCRs). For more information on programming these registers, refer to Section 10.3, "Chip Select Operation," and Section 7.3.1, "DRAM Controller Registers."

Figure 8-3 shows the byte lanes that external chip-select memory and DRAM should be connected to and the sequential transfers that would occur for each memory if a longword was transferred to it. A 16-bit memory should be connected to[31:16] of the MCF5251 data bus. For a longword transfer, the most significant word D[31:16] will be transferred on lane D[31:16], followed by the least significant word being transferred.

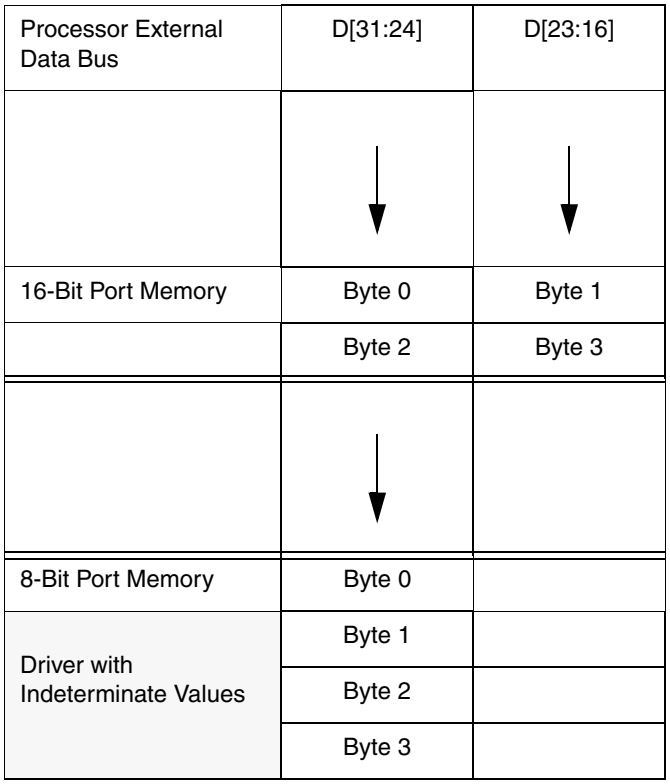


Figure 8-2. Connections for External Memory Port Sizes

8.5.1 Bus Cycle Execution

When a bus cycle is initiated, the processor compares the address of that bus cycle with the base address and mask configurations programmed for various memory-mapped peripherals. These include SRAM0, SRAM1, System Bus Controller 1 and 2, chip selects, and the DRAM. If no match is found, the cycle will

terminate in an error. If a match is found for any chip selects or DRAM, the bus cycle will be executed on the external bus. Chip select accesses follow timing diagrams given in this section. DRAM accesses are different. They are described in the section on the DRAM controller.

Table 8-5 shows the type of access as a function of match in various memory space programming registers.

Table 8-5. Accesses by Matches

KRAM Matches	SBC 2 Matches	SBC 1 Matches	Number of Chip Selects Register Matches	Number of DRAM Controller Register Matches	Type of Access
yes	any	any	any	any	on-chip SRAM
no	yes	any	any	any	SBC 2
no	no	yes	none	none	SBC 1
no	no	no	single	none	as defined by Chip-Select control register
no	no	no	none	single	as defined by DRAM control register
no	no	no	none	none	Undefined
All other combinations					Undefined

Basic operation of the MCF5251 bus is a three-clock bus cycle. During the first clock, the address is driven. $\overline{CSx}$ is asserted at the falling edge of the clock to indicate that address and attributes are valid and stable. Data and $\overline{TA}$ are sampled during the second clock of a bus-read cycle. $\overline{TA}$ is generated internally in the chip select module.

During a read, the external device provides data and is sampled at the rising edge at the end of the second bus clock. This data is concurrent with $\overline{TA}$, which is also sampled at the rising edge of the clock. During a write, the MCF5251 drives data from the rising clock edge at the end of the first clock to the rising clock edge at the end of the bus cycle.

Users can add wait states between the first and second clocks by delaying the assertion of $\overline{TA}$. This refers to internal transfers only and not the write cycles. This is done by programming the relevant chip select registers. If “0000” is programmed in the WS field of the relevant chip select register, a no wait cycle results. If *n* is programmed in the WS field, *n* wait cycles will result. The last clock of the bus cycle uses what would be an idle clock between cycles to provide hold time for address and write data. Figure 8-4 and Figure 8-6 show the basic read and write operations.

8.5.2 Read Cycle

The Read cycle as shown in Figure 8-3, will occur if the wait cycle field (WS) in the Chip Select Control Register (CSR) is programmed to value “0000”. The CS low time is increased with *n* clocks if *n* is programmed into the WS field.

During a read cycle, the MCF5251 receives data from memory or from a peripheral device. The read cycle flowchart is shown in Figure 8-3 while the read cycle timing diagram is shown in Figure 8-4.

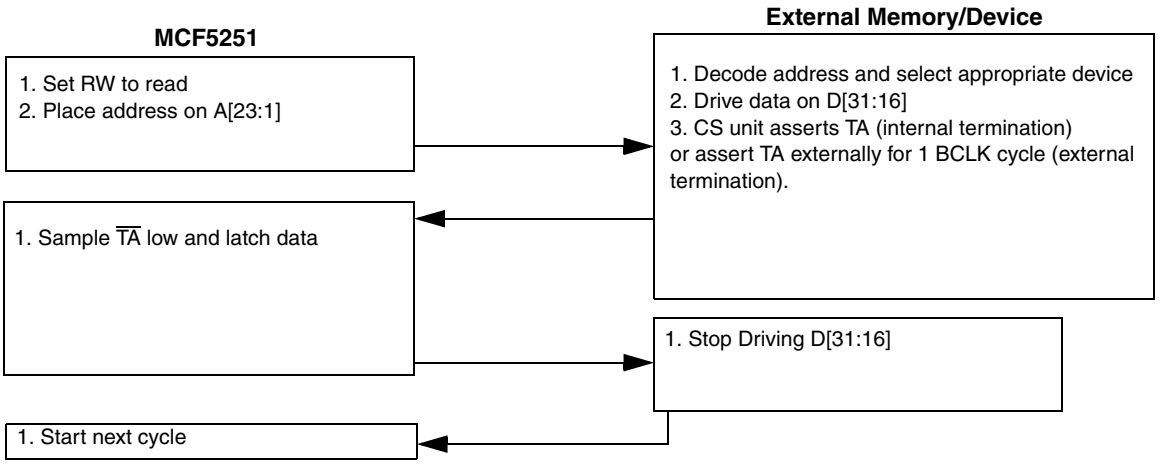


Figure 8-3. Read Cycle Flowchart

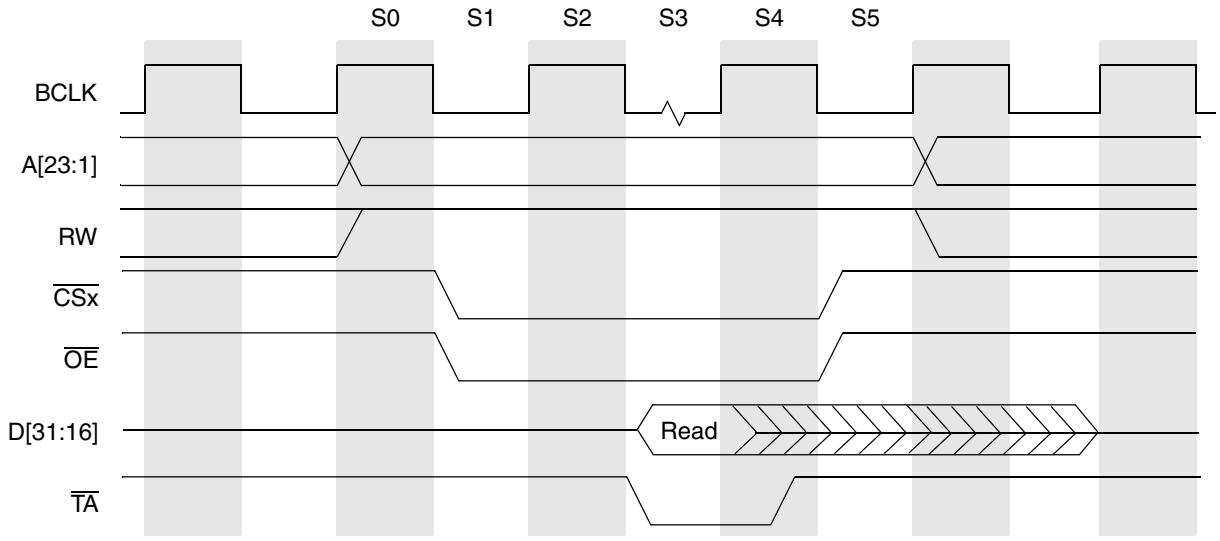


Figure 8-4. Basic Read Bus Cycle

A basic read bus cycle has six states (S0–S5). The signal timing relationship in the constituent states of a basic read cycle is as follows:

Table 8-6. Read Cycle States

State Name	Description[1,2]
STATE 0	The read cycle is initiated in state 0 (S0). On the rising edge of BCLK, the MCF5251 places a valid address on the address bus and drives RW high, if it is not already high.
STATE 1	The appropriate $\overline{CS}$ and $\overline{OE}$ are asserted on the falling edge of BCLK.
STATE 2	

Table 8-6. Read Cycle States (continued)

State Name	Description[1,2]
STATE 3	Data is made available by the external device and is sampled on the rising edge of BCLK with $\overline{TA}$ asserted. If $\overline{TA}$ not asserted before the rising edge of BCLK at the end of the first clock cycle, the MCF5251 inserts wait states (full clock cycles) until $\overline{TA}$ is asserted. If internal $\overline{TA}$ is requested (auto-acknowledge enabled in the chip select control register, CSCR) then $\overline{TA}$ is generated internally by the chip select module.
STATE 4	During state 4, $\overline{TA}$ should be negated by the external device or if auto-acknowledge is enabled will be negated internally by the chip select module.
STATE 5	$\overline{CS}$ and $\overline{OE}$ are negated on the falling edge of state 5 (S5). The MCF5251 stops driving the address lines and RW on the rising edge of BCLK, terminating the read cycle. The external device must have its drive from the bus. The external device must stop driving the bus. The rising edge of BCLK may be the start of state 0 for the next access cycle.

1 The external device has a maximum of 1.5 BCLK cycles after the start of S4 to three-state the data bus after data is sampled in S3 during a read cycle. This applies to basic read cycles and the last transfer of a burst.

2 The MCF5251 would not drive out data for a minimum of two BCLK cycles. However, another slave device may start driving the bus as soon as its chip select is asserted. Chip select may be asserted at the beginning of S1, so bus drive must stop before the end of S0. Under these conditions, data contention on the bus would not exist.

8.5.3 Write Cycle

The Write cycle as shown in Figure 8-6., will occur if the wait cycle field (WS) in the Chip Select Control Register (CSR) is programmed to value “0000”. The CS low time is increased with *n* clocks if *n* is programmed into the WS field.

During a write cycle, the MCF5251 sends data to the memory or to a peripheral device.

The write cycle flowchart is Figure 8-6. Write cycle timing diagram is Figure 8-7.

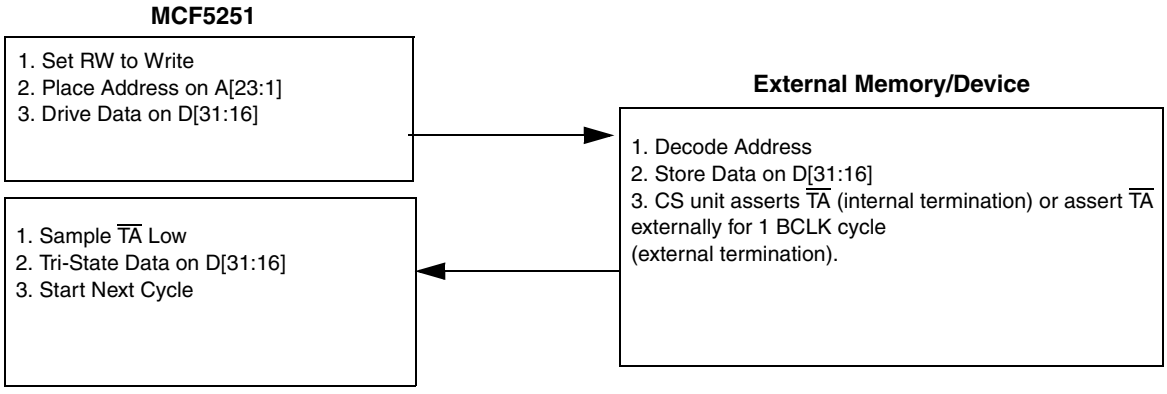


Figure 8-5. Write Cycle Flowchart

Figure 8-6 shows the description for the six states of a basic write cycle.

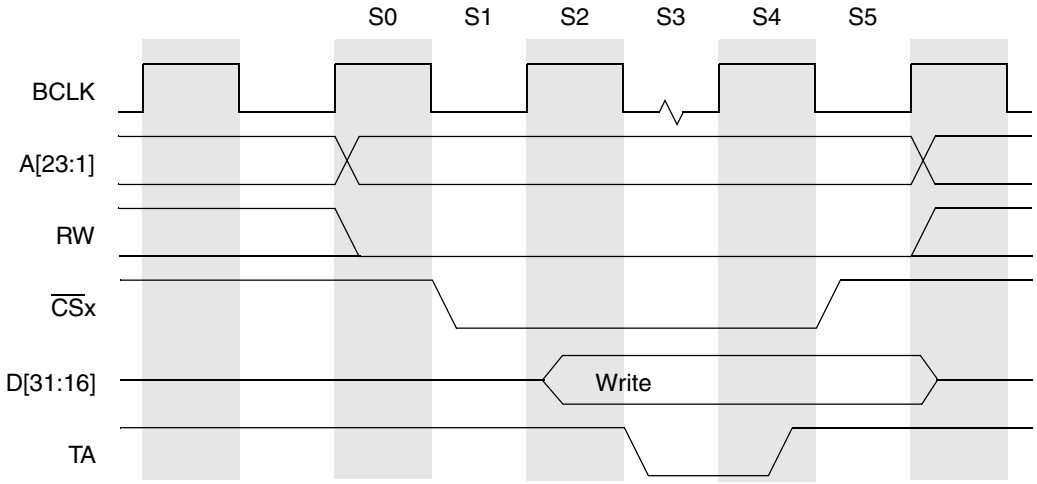


Figure 8-6. Basic Write Bus Cycle

Table 8-7. Write Cycle States

State Name	Description
STATE 0	The write cycle is initiated in state 0 (S0). On the rising edge of BCLK, the MCF5251 places a valid address on the address bus and drives RW low, if it is not already low.
STATE 1	The appropriate $\overline{CS}$ is asserted on the falling edge of BCLK.
STATE 2	The data bus is driven out of high impedance as data is placed on the bus on the rising edge of BCLK.
STATE 3	During state 3 (S3), the MCF5251 waits for a cycle termination signal ($\overline{TA}$). If $\overline{TA}$ is not asserted before the rising edge of BCLK at the end of the first clock cycle, the MCF5251 inserts wait states (full clock cycles) until $\overline{TA}$ is asserted. $\overline{TA}$ is generated internally by the chip select module. If internal $\overline{TA}$ is requested (auto-acknowledge enabled in the chip select control register, CSCR) then $\overline{TA}$ is generated internally by the chip select module.
STATE 4	During state 4, $\overline{TA}$ should be negated by the external device or if auto-acknowledge is enabled, negated internally by the chip select module.
STATE 5	$\overline{CS}$ is negated on the falling edge of BCLK in state 5 (S5). The MCF5251 stops driving the address lines and RW, terminating the write cycle. The data bus returns to high impedance on the rising edge of BCLK. The rising edge of BCLK may be the start of state 0 for the next access cycle.

8.5.4 Back-to-Back Bus Cycles

The MCF5251 can accommodate back-to-back bus cycles. The processor runs back-to-back bus cycles whenever possible. For example, when a longword read is started on a word-size bus, and burst read enable is disabled into the relevant chip select register, the processor will perform two word reads back to back. Figure 8-7 shows a read, followed by a write that occurs back to back.

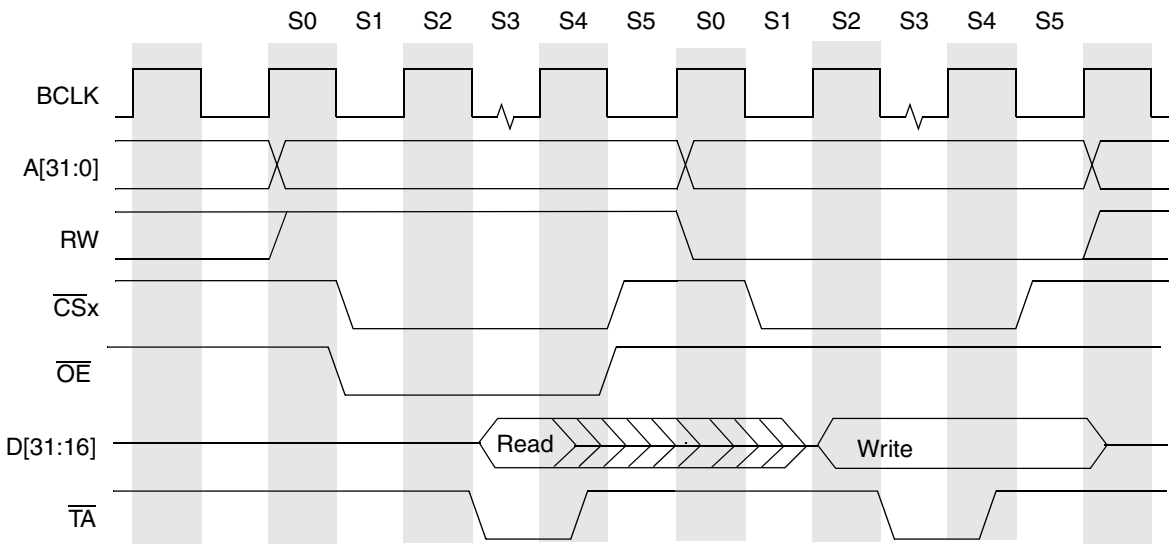


Figure 8-7. Back-to-Back Bus Cycle

8.5.5 Burst Cycles

When burst read enable or burst write enable is asserted into the relevant chip select register, the MCF5251 will initiate burst cycles any time a transfer size is larger than the port size the MCF5251 is transferring to. A line transfer to a 16-bit port would constitute a burst cycle of eight words of data.

The MCF5251 bus can support 3-2-2-2 burst cycles to maximize cache performance and optimize DMA transfers. Users can add wait states if desired by delaying termination of the cycle.

Through the chip select control registers, users can enable bursting on reads, bursting on writes or bursting on both reads and writes if desired.

8.5.5.1 Line Transfers

A line is defined as a 16-byte value, aligned in memory on 16-byte boundaries. Although the line itself is aligned on 16-byte boundaries, the line access does not necessarily begin on the aligned address.Therefore, the bus interface supports line transfers on multiple address boundaries. The allowable patterns during a line access are shown in Table 8-8.

Table 8-8. Allowable Line Access Patterns

Addr[3:2]	Longword Accesses
00	0 - 4 - 8 - C
01	4 - 8 - C - 0
10	8 - C - 0 - 4
11	C - 0 - 4 - 8

8.5.5.2 Line Read Bus Cycles

Figure 8-8 shows a line access read with zero wait states.

NOTE

The bus cycle begins similar to a basic read bus cycle with the first data transfer being sampled on the rising edge of S4. However, also notice that the next pipelined burst data is sampled one cycle later on the rising edge of S6. Each subsequent pipelined data burst will be single cycle until the last cycle which can be held for a maximum of 2 BCLK past the $\overline{TA}$ assertion. $\overline{CS}$ and $\overline{OE}$ remain asserted throughout the burst transfer.

Figure 8-8 shows a line access read with one wait state. Wait states can be programmed in the chip select control register (CSCRs) to give the peripheral or memory more time to return read data. This figure follows the same execution as a zero-wait state read burst with the exception of an added wait state.

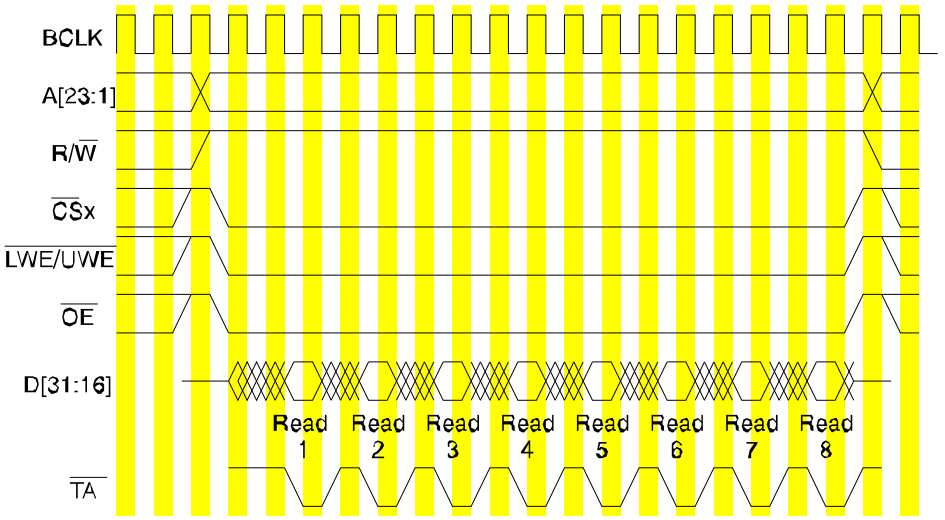


Figure 8-8. Line Read Burst (one wait cycle)

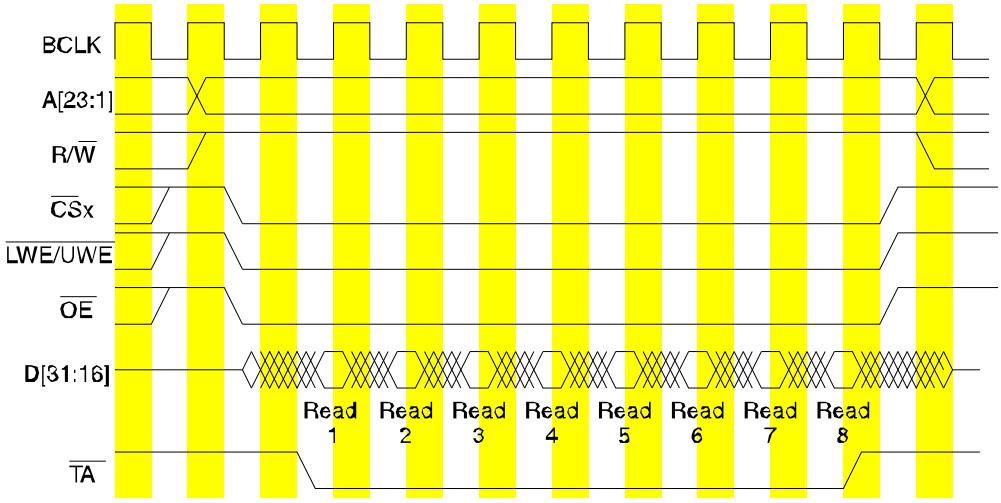


Figure 8-9. Line Read Burst (no wait cycles) Line Write Bus Cycles

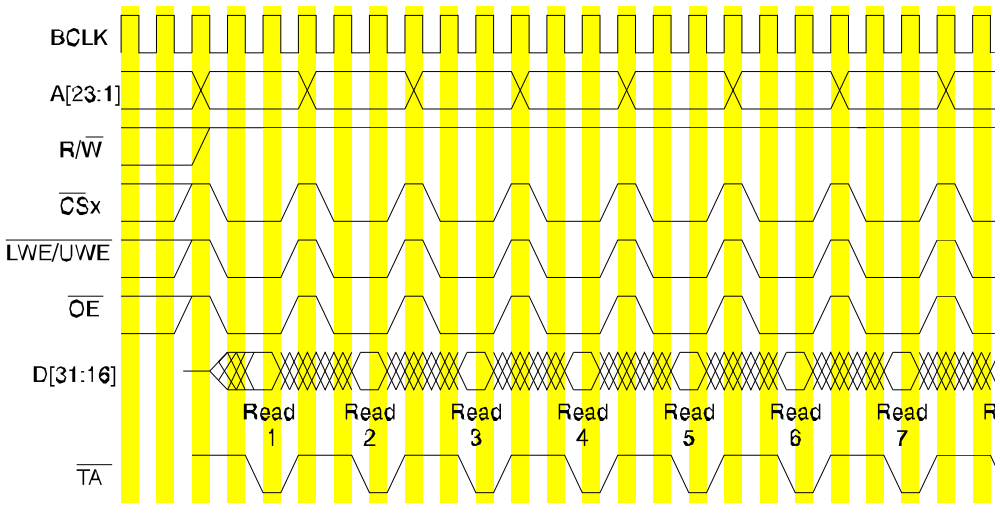


Figure 8-10. Line Read Burst-Inhibited

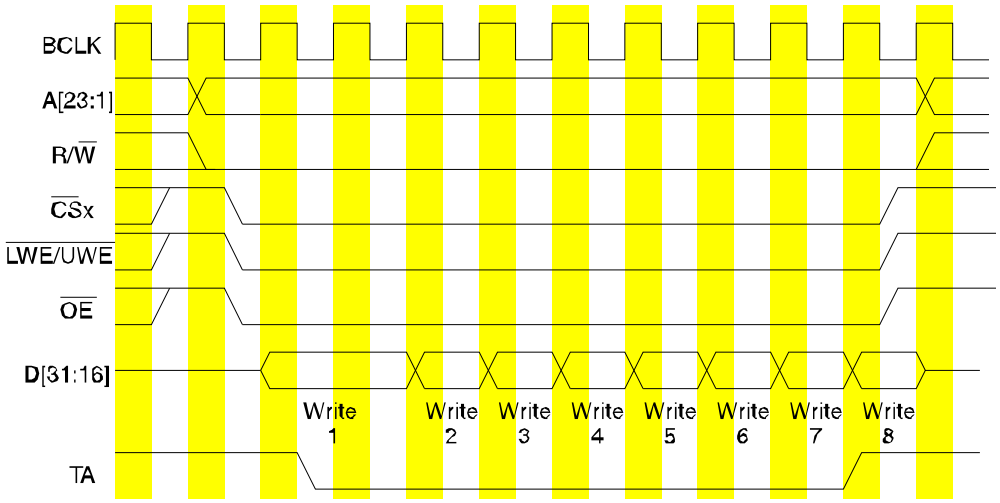


Figure 8-11. Line Write Burst (no wait cycles)

NOTE

The bus cycle begins similar to a basic write bus cycle with data being driven one clock after the address. Also notice that the next pipelined burst data is driven one cycle after the write data has been registered (on the rising edge of S6). Each subsequent pipelined write data burst will be a single cycle. $\overline{CS}$ remains asserted throughout the burst transfer.

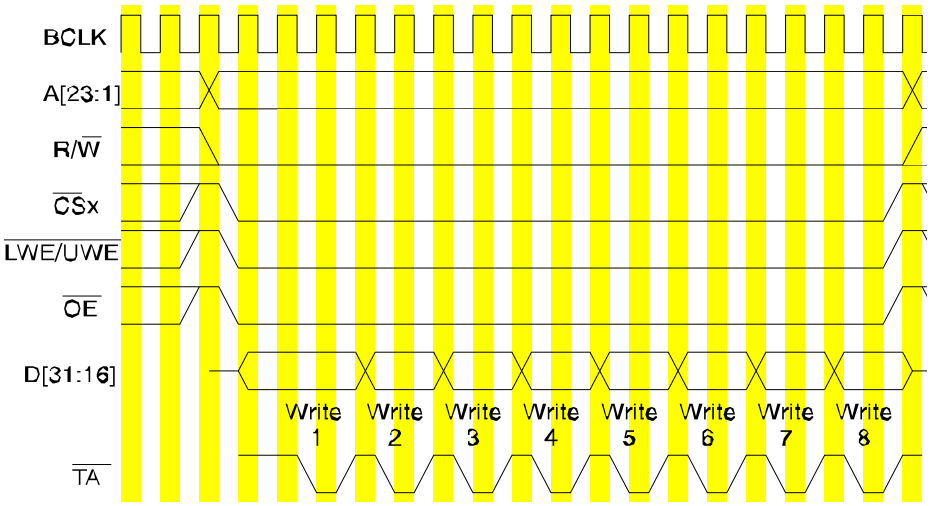


Figure 8-12. Line Write Burst with One Wait State

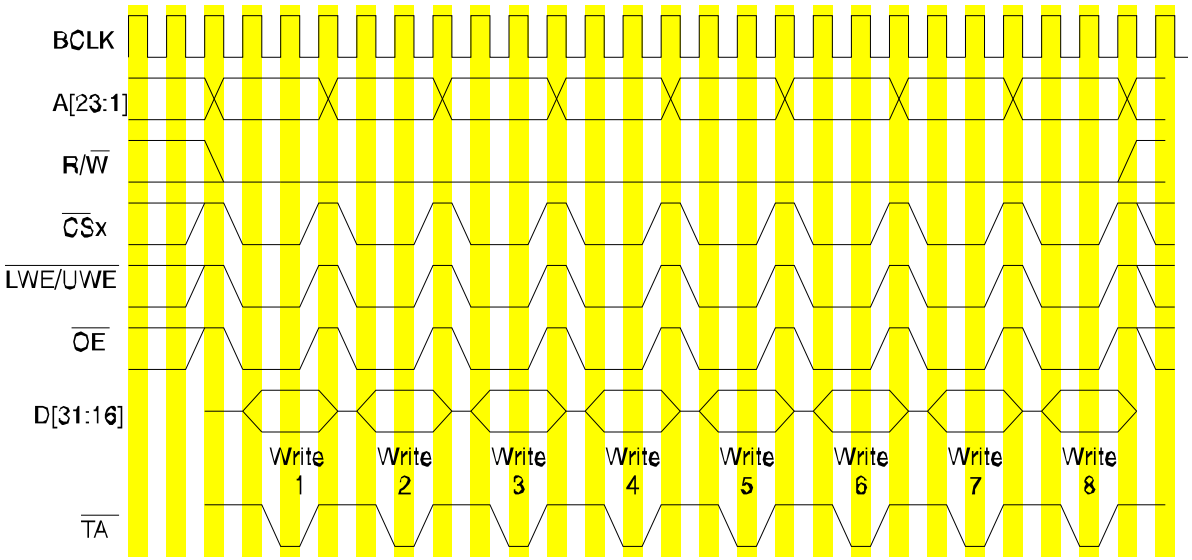


Figure 8-13. Line Write Burst-Inhibited

8.6 Misaligned Operands

All MCF5251 data formats can be located in memory on any byte boundary. A byte operand is properly aligned at any address; a word operand is misaligned at an odd address; and a longword is misaligned at an address that is not evenly divisible by four. Unlike opcodes, because operands can reside at any byte boundary, they are allowed to be misaligned. Although the MCF5251 does not enforce any alignment restrictions for data operands (including program counter (PC) relative data addressing), some performance degradation occurs when additional bus cycles are required for longword or word operands that are misaligned. For maximum performance, data items should be aligned on their natural boundaries. All instruction words and extension words (opcodes) must reside on word boundaries. An address error exception will occur with any attempt to prefetch an instruction word at an odd address.

The MCF5251 converts misaligned operand accesses that are noncachable to a sequence of aligned accesses. Figure 8-14 illustrates the transfer of a longword operand from a byte address to a 32-bit port, requiring more than one bus cycle. The slave device supplies the byte and acknowledges the data transfer. The next two bytes are transferred during the second cycle. During the third cycle, the byte offset is now $0; the port supplies the final byte and the operation is complete. Figure 8-14 is similar to the example illustrated in Figure 8-15 except that the operand is word-sized and the transfer requires only two bus cycles.

	31	24 23	16 15	8 7	0
TRANSFER 1	–	OP 3	–	–	
TRANSFER 2	–	–	OP 2	OP 1	
TRANSFER 3	OP 0	–	–	–	

Figure 8-14. Misaligned Longword Transfer

	31	24 23	16 15	8 7	0
TRANSFER 1	–	–	–	OP 1	
TRANSFER 2	OP 0	–	–	–	

Figure 8-15. Misaligned Word Transfer

8.7 Reset Operation

The MCF5251 processor supports one type of reset which resets the entire MCF5251: the external master reset input ($\overline{\text{RSTI}}$).

To perform a master reset, an external device asserts the reset input pin ($\overline{\text{RSTI}}$). When power is applied to the system, external circuitry should assert $\overline{\text{RSTI}}$ for a minimum of 16 CRIN cycles after Vcc is within tolerance. Figure 8-16 is a functional timing diagram of the master reset operation, illustrating relationships among V_{DD}, $\overline{\text{RSTI}}$, mode selects, and bus signals. The crystal oscillation on CRIN, CROUT must be stable by the time V_{DD} reaches the minimum operating specification. The crystal should start oscillating as V_{DD} is ramped up to clear out contention internal to the MCF5251 processor caused by the random states of internal flip-flops on power up. $\overline{\text{RSTI}}$ is internally synchronized for two CRIN cycles before being used and must meet the specified setup and hold times in relationship to CRIN to be recognized.

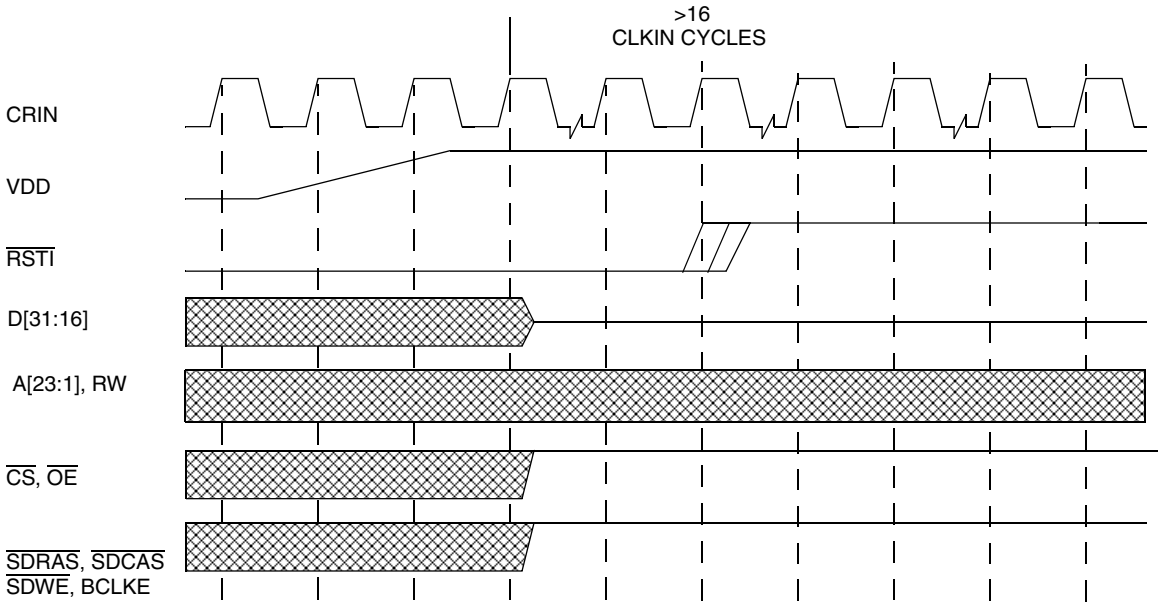


Figure 8-16. Master Reset Timing

During the master reset period, the data bus is being tri-stated, the address bus is driven to any value, and all other bus signals are driven to their negated state. Once RSTI negates, the bus stays in this state until the core begins the first bus cycle for reset exception processing. A master reset causes any bus cycle (including DRAM refresh cycle) to terminate. In addition, master reset initializes registers appropriately for a reset exception.

If at power-on reset, the MCF5251 is configured to boot from external memory connected to CS0. Then CS0 is configured to address the external boot ROM / Flash. The configuration for CS0 at this time is hard-wired inside the MCF5251.

Configuration is summarized in Table 8-9.

Table 8-9. Power-On Reset Configuration for CS0

Port Size	16 Bits
Cycle type	Internal termination, 15 wait cycles Burst inhibit asserted for both read and write cycles

8.7.1 Software Watchdog Reset

The software watchdog reset is performed anytime the executing software does not provide the correct write sequence with the enable-control bit set. This reset helps recovery from runaway software or nonterminated bus cycles. During the software watchdog reset period all signals are driven either to a high impedance state or a negated state as appropriate.

Chapter 9
System Integration Module (SIM)

9.1 SIM Overview

This chapter describes the operation, memory map and register descriptions of the System Integration Module (SIM) registers, including the interrupt controller and system-protection functions for the MCF5251 processor. The SIM provides overall control of the internal and external buses and serves as the interface between the ColdFire® core and the internal peripherals or external devices. The SIM also configures the general purpose input/output and enables the CPU HALT instruction.

9.1.1 SIM Features

- Module Base Address Registers (MBAR and MBAR2)
  - — Base address location of all internal peripherals and SIM resources
  - — Address space masking to internal peripherals and SIM resources
- Interrupt Controller
  - — Two interrupt controllers
  - — Programmable interrupt level (1–7) for peripheral interrupts
- System Protection and Reset Status
  - — Reset status to indicate cause of last reset
  - — Software watchdog timer with optional secondary bus monitor functionality
- Bus Arbitration Control Register (MPARK)
  - — Enables display of internal accesses on the external bus for debug
- General purpose input/output registers
  - — Defines general-purpose inputs and outputs
  - — Edge interrupt triggers on general-purpose I/Os, 0 to 6
- Software interrupts
  - — Allow programmer to make interrupt pending under software control

9.2 SIM Memory Map and Register Definitions

This chapter provides the SIM register memory map, programming and configuration register descriptions, interrupt interface register descriptions, secondary interrupt controller register descriptions, and software interrupts.

9.2.1 SIM Register Memory Map

Table 9-2 shows the memory map of all the SIM registers. The internal registers in the SIM are memory-mapped registers offset from the MBAR or MBAR2 address pointers. The following should be noted when programming the MBAR registers:

- The Module Base Address Registers are accessed in supervisor mode only using the MOVEC instruction.
- The MBAR and MBAR2 are accessible using the debug module as read/write registers. See Chapter 20, "Background Debug Mode (BDM) Interface," for more details.

Table 9-1. MBAR Register Addresses

Address	Name	Size (Bytes)	Description
CPU + $C0F	MBAR	4	Module base address register
CPU + $C0E	MBAR2	4	Module base address register 2

Table 9-2. SIM Memory Map

Address	Description	0	1	2	3
MBAR + $000	SYSTEM CONTROL REG	RSR	SYPCR	SWIVR	SWSR
MBAR + $004	–	Reserved			
MBAR + $008	–	Reserved			
MBAR + $00C	BUS MASTER CONTROL REG	MPARK	Reserved		
MBAR + $010	–	Reserved			
MBAR + $014	–				
MBAR + $018	–				
MBAR + $01C	–				
MBAR + $020	–				
MBAR + $024	–				
MBAR + $028	–				
MBAR + $02C	–				
MBAR + $030	–				
MBAR + $034	–				
MBAR + $038	–				
MBAR + $03C	–				
MBAR + $040	Primary interrupt Pending Reg	IPR			
MBAR + $044	Primary Interrupt Mask Reg	IMR			
MBAR + $04C	Primary Interrupt Control Reg	ICR0	ICR1	ICR2	ICR3
MBAR + $050	Primary Interrupt Control Reg	ICR4	ICR5	ICR6	ICR7

Table 9-2. SIM Memory Map (continued)

Address	Description	0	1	2	3
MBAR + $054	Primary Interrupt Control Reg	ICR8	ICR9	ICR10	ICR11
MBAR2 + $000	GPIO 0–31 input reg	GPIO-READ (READ ONLY)			
MBAR2 + $004	GPIO 0–31 output reg	GPIO-OUT			
MBAR2 + $008	GPIO 0–31 output enable reg	GPIO-ENABLE			
MBAR2 + $00C	GPIO 0–31 function select	GPIO-FUNCTION			
MBAR + $0AC	DeviceID Reg	–			
MBAR2 + $0B0	GPIO 32–63 input reg	GPIO1-READ (READ ONLY)			
MBAR2 + $0B4	GPIO 32–63 output reg	GPIO1-OUT			
MBAR2 + $0B8	GPIO 32–63 output enable reg	GPIO1-ENABLE			
MBAR2 + $0BC	GPIO 32–63 function select	GPIO1-FUNCTION			
MBAR2 + $140	Secondary interrupts 0–7 priority	INTPRI1			
MBAR2 + $144	Secondary interrupts 8–15 priority	INTPRI2			
MBAR2 + $148	Secondary interrupts 16–23 priority	INTPRI3			
MBAR2 + $14C	Secondary interrupts 24–31 priority	INTPRI4			
MBAR2 + $150	Secondary interrupts 32–39 priority	INTPRI5			
MBAR2 + $154	Secondary interrupts 40–47 priority	INTPRI6			
MBAR2 + $158	Secondary interrupts 48–55 priority	INTPRI7			
MBAR2 + $15C	Secondary interrupts 56–63 priority	INTPRI8			
MBAR2 + $164	Spurious secondary interrupt vector	SPURVEC			
MBAR2 + $168	Secondary interrupt base vector register	INTBASE			
MBAR2 + $198	Software interrupts and interrupt monitor	EXTRAINT			

9.3 SIM Module Programming Registers

This section provides the Module Base Address, DeviceID, and Interrupt Controller registers and their descriptions.

9.3.1 Module Base Address Registers

The base address of all internal peripherals is determined by the MBAR and MBAR2 registers.

The MBAR and MBAR2 are 32-bit write-only supervisor control registers that physically reside in the SIM. They are accessed in the CPU address spaces $C0F and $C0E using the MOVEC instruction. Refer to the *ColdFire Family Programmer's Reference Manual* for use of MOVEC instruction. The MBAR and MBAR2 can be read when in debug mode using background debug commands.

At system reset, the MBAR valid bits (MBAR[0], MBAR2[0]) are cleared to prevent incorrect reference to resources before the MBAR or MBAR2 are written. The remainder of the MBAR and MBAR2 bits are

uninitialized. To access the MBAR and MBAR2 peripherals, users should write MBAR and MBAR2 with the appropriate base address and set the valid bit after system reset.

The MBAR2 base address defines a single relocatable memory block on any 1024-Mbyte boundary. If the MBAR2 valid bit is set, the base address field is compared to the upper two bits of the full 32-bit internal address to determine if an MBAR2 peripheral is being accessed.

Any processor bus access is first compared for SRAM match (RAMBAR registers), then it is compared against MBAR and MBAR2. If no match is found in any of these registers, the cycle will be mapped to the Chip Select and SDRAM units.

Table 9-1 shows the bits in the module base address register (MBAR), and Table 9-2 shows the bits in the MBAR2.

Address CPU + $C0F — Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	BA31	BA30	BA29	BA28	BA27	BA26	BA25	BA24	BA23	BA22	BA21	BA20	BA19	BA18	BA17	BA16
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	BA15	BA14	BA13	BA12				WP		AM	C/I	SC	SD	UC	UD	V
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0

Figure 9-1. Module Base Address Register (MBAR)

Table 9-3. Module Base Address Register (MBAR) Field Descriptions

Field	Description
31–12 BA	The Base Address field defines the base address for a minimum 4 Kbyte address range.
11–9	Reserved.
8 WP	The Write Protect bit is the mask bit for write cycles in the MBAR-mapped register address range. 0 Module address range is read/write 1 Module address range is read only
7	Reserved.
6 AM	AM–Alternate Master Mask. When AM = 0 and an alternate master actually accesses the MBAR-mapped registers; bits SC, SD, UC, and UD (MBAR[4:1]) are “don’t cares” in the address decoding. 0 Alternate master access allowed 1 Alternate master access masked
5 C/I	Mask CPU Space and Interrupt Acknowledge Cycle. 0 IACK cycle mapped to MBAR space 1 IACK cycle not responded to by MBAR peripherals

Table 9-3. Module Base Address Register (MBAR) Field Descriptions (continued)

Field	Description
4 SC	Mask Supervisor Code space in MBAR address range. 0 Supervisor code access allowed 1 Supervisor code access masked
3 SD	Mask Supervisor Data space in MBAR address range. 0 Supervisor data access allowed 1 Supervisor data access masked
2 UC	Mask User Code space in MBAR address range. 0 User code access allowed 1 User code access masked
1 UD	Mask User Data space in MBAR address range. 0 User data space access allowed 1 User data space access masked
0 V	This bit defines when the base address is valid: 0 MBAR address space not visible by CPU 1 MBAR address space visible by CPU

The following example shows how to set the MBAR to location $10000000 using the D0 register. A “1” in the least significant bit validates the MBAR location. This example assumes that all accesses are valid:

```
move.l #$10000001,DO
movec DO,MBAR
```

Address CPU + $C0E — Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BA31	BA30														
W																
Reset	0	0	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R									LS7	LS6	LS5	LS4	LS3	LS2	LS1	V
W																
Reset	–	–	–	–	–	–	–	–	0	0	0	0	0	0	0	0

Figure 9-2. Second Module Base Address Register (MBAR2)

Table 9-4. Second Module Base Address Register (MBAR2)Field Descriptions

Field	Description
31–30 BA	The Base Address field defines the base address for a 1024 Mbyte address range. If V-bit in MBAR2 is set, address range Base Address to BaseAddress + $3FFF FFFF are mapped to MBAR2 space, and cannot be used for MBAR, SDRAM or Chip Select.
29–8	Reserved, should be cleared.

Table 9-4. Second Module Base Address Register (MBAR2)Field Descriptions (continued)

Field	Description
7–1 LS	If interrupts in both the “primary” and the “secondary” interrupt controllers have the same interrupt level pending then bits LS[7:1] determine which interrupt controller gets priority. If the bit is cleared, the primary interrupt controller gets priority. If the bit is set, the secondary interrupt controller gets priority. There are 7 LSn bits, one for each interrupt level.
0 V	The Valid bit defines if the CPU can access the MBAR2 mapped peripherals. 0 MBAR2 address space not visible by CPU 1 MBAR2 address space visible by CPU

9.3.2 Device ID Register

The DeviceID register is a read only register that allows the software to determine what hardware it is running on. The register contains the part number in the upper 24 bits, the mask revision number in the lower 8 bits, and is read as 0x005251rr, where *rr* is the revision number.

This register allows developers the flexibility to write code to run on more than one device. The revision number allows developers to distinguish between different mask versions that may have minor changes or bug fixes. For example, developers may want to distribute a single code image or library for use on different revisions of the silicon.

Address MBAR2 + 0xAC — Access: User read

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Part Number								Mask Revision							
W																
Reset	0	1	0	1	0	0	0	1	–	–	–	–	–	–	–	0

Figure 9-3. DeviceID Register (DeviceID)

9.4 Interrupt Interface Registers

For legacy reasons, there are two interrupt controllers on the MCF5251. This section provides the programming of the two interrupt controller registers and their register descriptions.

The primary interrupt controller is centralized, and services the following:

- Software Watchdog Timer (SWT)
- Timer modules
- I^2C0 module

- UART0 and UART1 modules
- DMA modules
- QSPI module

The secondary interrupt controller is decentralized, and services the following:

- GPIO interrupts
- Audio interface module
- MemoryStick/SD module
- AD convertor module
- I^2C1 module
- Software triggered Interrupts
- FlexCAN modules
- USB module
- ATA module
- UART2 module

9.4.1 Primary Interrupt Controller Registers

Primary internal interrupt sources have their own interrupt control registers ICR[11:0], IPR, and IMR. Table 9-5 gives the location and description of each ICR.

Table 9-5. Primary Interrupt Control Register Memory Map

Address	Name	Width	Description	Reset Value	Access
MBAR + $04C	ICR0	8	SWT	$00	R/W
MBAR + $04D	ICR1	8	TIMER 0	$00	R/W
MBAR + $04E	ICR2	8	TIMER 1	$00	R/W
MBAR + $04F	ICR3	8	I^2C0	$00	R/W
MBAR + $050	ICR4	8	UART 0	$00	R/W
MBAR + $051	ICR5	8	UART 1	$00	R/W
MBAR + $052	ICR6	8	DMA 0	$00	R/W
MBAR + $053	ICR7	8	DMA 1	$00	R/W
MBAR + $054	ICR8	8	DMA 2	$00	R/W
MBAR + $055	ICR9	8	DMA 3	$00	R/W
MBAR + $056	ICR10	8	QSPI	$00	R/W
MBAR + $057	ICR11	8	Reserved	–	–

Primary interrupts are programmed to a level and priority. All primary interrupts have a unique Interrupt Control Register (ICR). There are 28 possible priority levels, for the primary interrupts.

Address *See Memory Map Table 9-5* Access: User read/write

	7	6	5	4	3	2	1	0
R	AVEC			IL[2]	IL[1]	IL[0]	IP[1]	IP[0]
W								
Reset	0	–	–	0	0	0	0	0

Figure 9-4. Interrupt Control Register (ICR)

Table 9-6. Interrupt Control Register (ICR) Field Descriptions

Field	Description
7 AVEC	The Autovector Enable bit determines whether the interrupt-acknowledge cycle requires an autovector response (for the internal interrupt level indicated in IL[2:0] for each interrupt). 0 Interrupting source returns vector during interrupt-acknowledge cycle 1 SIM generates auto vector during interrupt acknowledge cycle
6–5	Reserved.
4–2 IL	The Interrupt Level bits indicate the interrupt level assigned to each interrupt input.
1–0 IP	The Interrupt Priority bits indicate the interrupt priority within the interrupt level assignment. Table 9-7 shows the priority levels associated with the IP contents.

Table 9-7. Interrupt Priority Assignment

IP[1:0]	Priority
00	Lower
01	Low
10	High
11	Higher

Table 9-8 shows all possible primary source priority schemes for the MCF5251. The interrupt source in this table can be any internal interrupt source programmed to the given level and priority. For example, assume that two internal interrupt sources were programmed to IL[2:0] =110, one having a priority of IP[1:0] = 01 and one having a priority of IP[1:0] = 10. If both assert an interrupt request at the same time, the order of servicing would occur as follows:

1. Internal module with IL[2:0] =110 and IP[1:0] = 10 would be serviced first
2. Internal module with IL[2:0] = 110 and IP[1:0] = 01 would be serviced last

Table 9-8. Interrupt Priority Scheme

Interrupt Level	Internal Module ICR Reg			Interrupt Source
	IL[2:0]	IP[1]	IP[0]	
7	111	1	1	Internal Module
7	111	1	0	Internal Module

Table 9-8. Interrupt Priority Scheme (continued)

Interrupt Level	Internal Module ICR Reg			Interrupt Source
	IL[2:0]	IP[1]	IP[0]	
7	111	0	1	Internal Module
7	111	0	0	Internal Module
6	110	1	1	Internal Module
6	110	1	0	Internal Module
6	110	0	1	Internal Module
6	110	0	0	Internal Module
5	101	1	1	Internal Module
5	101	1	0	Internal Module
5	101	0	1	Internal Module
5	101	0	0	Internal Module
4	100	1	1	Internal Module
4	100	1	0	Internal Module
4	100	0	1	Internal Module
4	100	0	0	Internal Module
3	011	1	1	Internal Module
3	011	1	0	Internal Module
3	011	0	1	Internal Module
3	011	0	0	Internal Module
2	010	1	1	Internal Module
2	010	1	0	Internal Module
2	010	0	1	Internal Module
2	010	0	0	Internal Module
1	001	1	1	Internal Module
1	001	1	0	Internal Module
1	001	0	1	Internal Module
1	001	0	0	Internal Module

NOTE

Multiple internal modules should not be assigned to the same interrupt level and same interrupt priority when configuring the ICR registers. This can cause erratic chip behavior.

9.4.1.1 Interrupt Mask Register

The IMR register is used to mask both internal and external interrupt sources from occurring.

Address MBAR + 0x44 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R														QSPI	DMA3	DMA2
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	1	1	1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DMA1	DMA0	UART1	UART0	I^2C0	TIMER1	TIMER0	SWT								
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	–

Figure 9-5. Interrupt Mask Register (IMR)

Table 9-9. Interrupt Mask Register (IMR) Field Descriptions

Field	Description
31–18	Reserved.
17–8 IMR	Each Interrupt Mask bit corresponds to an interrupt source defined in the Interrupt Control Register (ICR). An interrupt is masked by setting the corresponding bit in the IMR. When a masked interrupt occurs, the corresponding bit in the IPR is still set, regardless of the setting of the IMR bit, but no interrupt request is passed to the core processor. At system reset, all defined bits are initialized high, thereby masking all interrupts. The proper procedure for masking interrupt sources is to first set the core's status register interrupt mask level to the level of the source being masked in the IMR. Then, the IMR bit can be masked. An interrupt can be masked by setting the corresponding bit in the IMR and enable an interrupt by clearing the corresponding bit in the IMR.
7–0	Reserved.

9.4.1.2 Interrupt Pending Register

The IPR makes visible the interrupt sources that have an interrupt pending.

Address MBAR + 0x40 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R														QSPI	DMA3	DMA2
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DMA1	DMA0	UAR1	UAR0	I^2C	TIMER1	TIMER0	SWT								
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 9-6. Interrupt Pending Register (IPR)

Table 9-10. Interrupt Pending Register (IPR) Field Descriptions

Field	Description
31–19	Reserved.
18–8 IPR	Each Interrupt Pending bit corresponds to an interrupt source defined by the Interrupt Control Register. At every clock this register samples the signal generated by the interrupting source. The corresponding bit in this register reflects the state of the interrupt signal even if the corresponding mask bit were set. The IPR is a read-only longword register. 0 The corresponding interrupt source does not have an interrupt pending 1 The corresponding interrupt source has an interrupt pending
7–0	Reserved, should be cleared.

9.4.2 Secondary Interrupt Controller Registers

The secondary controller serves 64 interrupt sources with programmable interrupt levels. All 64 interrupts are auto-vectored. Interrupt pending registers and interrupt mask registers are decentralized, and available in the modules that own the interrupts.

Table 9-11. Secondary Interrupt Controller Registers Memory Map

Address	Name	Width	Description	Reset Value	Access
MBAR2 + $140	INTPRI1	32	Interrupts 0–7 priority	$00	R/W
MBAR2 + $144	INTPRI2	32	Interrupts 8–15 priority	$00	R/W
MBAR2 + $148	INTPRI3	32	Interrupts 16–23 priority	$00	R/W
MBAR2 + $14C	INTPRI4	32	Interrupts 24–31 priority	$00	R/W
MBAR2 + $150	INTPRI5	32	Interrupts 32–39 priority	$00	R/W
MBAR2 + $154	INTPRI6	32	Interrupts 40–47 priority	$00	R/W
MBAR2 + $158	INTPRI7	32	Interrupts 48–55 priority	$00	R/W
MBAR2 + $15C	INTPRI8	32	Interrupts 56–63 priority	$00	R/W

Table 9-11. Secondary Interrupt Controller Registers Memory Map (continued)

Address	Name	Width	Description	Reset Value	Access
MBAR2 + $16B	INTBASE	8	Interrupt base vector	$00	R/W
MBAR2 + $167	SPURVEC	8	spurious vector	$00	R/W

9.4.2.1 Interrupt Level Selection

The interrupt level, intpri[1:7], of the 64 interrupts serviced by the secondary interrupt controller can be programmed for every interrupt separately. Every interrupt is given a 4-bit field in one of the interrupt priority register. This 4-bit field controls level setting for the interrupt. Values 1–7 correspond with ColdFire interrupt priorities. Value 0 is off.

Table 9-12. Secondary Interrupt Level Programming Bit Assignment

Address	Name	Bit 31–28	Bit 27–24	Bit 23–20	Bit 19–16	Bit 15–12	Bit 11–8	Bit 7–4	Bit 3–0
MBAR2 + $140	INTPRI1	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0
MBAR2 + $144	INTPRI2	INT15	INT14	INT13	INT12	INT11	INT10	INT9	INT8
MBAR2 + $148	INTPRI3	INT23	INT22	INT21	INT20	INT19	INT18	INT17	INT16
MBAR2 + $14C	INTPRI4	INT31	INT30	INT29	INT28	INT27	INT26	INT25	INT24
MBAR2 + $150	INTPRI5	INT39	INT38	INT37	INT36	INT35	INT34	INT33	INT32
MBAR2 + $154	INTPRI6	INT47	INT46	INT45	INT44	INT43	INT42	INT41	INT40
MBAR2 + $158	INTPRI7	INT55	INT54	INT53	INT52	INT51	INT50	INT49	INT48
MBAR2 + $15C	INTPRI8	INT63	INT62	INT61	INT60	INT59	INT58	INT57	INT56

9.4.2.2 Interrupt Vector Generation Register

All secondary interrupts are autovectored. The vector number for interrupt 0 is given by register INTBASE. The vector numbers for the other interrupts are offset from this number. Vector number for interrupt 23 is e.g. INTBASE + *23*. The secondary interrupt controller will generate vector numbers INTBASE to INTBASE + *63* for its 64 interrupts.

Address MBAR2 + $16B — Access: User read/write

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

Figure 9-7. INTBase Register

Table 9-13. INTBase Register Field Descriptions

Field	Description
7–0 BASE	This is the 8-bit interrupt vector for interrupt 0. Vector numbers for other interrupts are obtained by adding the interrupt number to BASE. For example: Interrupt 23 vector is base + 23.

9.4.2.3 Spurious Vector Register

The SPURVEC register contains the interrupt vector number that is fed when a spurious interrupt event occurs on the secondary interrupt controller. A spurious interrupt occurs when a pending interrupt causes the ColdFire processor to feed an interrupt vector, but before the interrupt vector can be fed, the pending interrupt disappears.

Address MBAR2 + $167 Access: User read/write

	7	6	5	4	3	2	1	0
R W	spurvec[7]	spurvec[6]	spurvec[5]	spurvec[4]	spurvec[3]	spurvec[2]	spurvec[1]	spurvec[0]
Reset	–	–	–	–	–	–	–	–

Figure 9-8. Spurvec Register

9.4.2.4 Secondary Interrupt Sources

The 64 secondary interrupts used by the MCF5251 modules are provided in Table 9-14.

Table 9-14. Secondary Interrupt Sources

Interrupt	Interrupt Name	Module	Description
63	A/D	A/D	A to D convertor
62	IIC1	IIC1	iic1 interrupt
61	IPADDRESSERROR	SIM	IP address error cycle interrupt
60	See Table 9-15	FLASHINTER	SD/MemoryStick interrupt
59	See Table 9-15	FLASHINTER	SD/MemoryStick interrupt
58	See Table 9-15	FLASHINTER	SD/MemoryStick interrupt
57	See Table 9-15	FLASHINTER	SD/MemoryStick interrupt
56	CDROMNEWBLOCK	AUDIO	CD-ROM new block interrupt
55	CDROMILSYNC	AUDIO	CD-ROM ilsync interrupt
54	CDROMNOSYNC	AUDIO	CD-ROM nosync interrupt
53	CDROMCRCERR	AUDIO	CD-ROM crc error interrupt
52	USB	USB	USB controller
51	ATA	ATA	ATA module
50	SOFTINT3	AUXINT	Software interrupt 3

Table 9-14. Secondary Interrupt Sources (continued)

Interrupt	Interrupt Name	Module	Description
49	SOFTINT2	AUXINT	Software interrupt 2
48	SOFTINT1	AUXINT	Software interrupt 1
47	SOFTINT0	AUXINT	Software interrupt 0
46	CAN0MESSAGE	FLEXCAN0	Message buffer handler
45	CAN0STATUS	FLEXCAN0	Error, Wakeup, Busoff interrupts
44	CAN1MESSAGE	FLEXCAN1	Message buffer handler
43	CAN1STATUS	FLEXCAN1	Error, Wakeup, Busoff interrupts
42	UART2	UART2	3rd UART interrupt
41	ATAUSBDMA	ATADMA	Special DMA channel intended for use with 16K SRAM block associated with the USB and ATA modules
40	USBWUP	USBPWR	USB Wakeup
39	GPI7 (Not applicable, no GPI7 pin)	SIM	gpio interrupt
38	GPI6	SIM	gpio interrupt
37	GPI5	SIM	gpio interrupt
36	GPI4	SIM	gpio interrupt
35	GPI3	SIM	gpio interrupt
34	GPI2	SIM	gpio interrupt
33	GPI1	SIM	gpio interrupt
32	GPI0	SIM	gpio interrupt
31	IIS1TXUNOV	AUDIO	iis1 transmit FIFO under / over
30	IIS1TXRESYN	AUDIO	iis1 transmit FIFO resync
29	IIS2TXUNOV	AUDIO	iis2 transmit FIFO under / over
28	IIS2TXRESYN	AUDIO	iis2 transmit FIFO resync
27	EBUTXUNOV	AUDIO	IEC958 transmit FIFO under / over
26	EBUTXRESYN	AUDIO	IEC958 transmit FIFO resync
25	IEC958-1 CNEW	AUDIO	IEC958-1 receives new C control channel frame
24	IEC958-1 VALNOGOOD	AUDIO	IEC958 validity flag no good
23	IEC958-1 PARITY OR SYMBOL ERROR	AUDIO	IEC958 receiver 1 bit or symbol error
22	PDIR3UNOV	AUDIO	Processor data in 3 under/over
21	UCHANTXEMPTY	AUDIO	U channel transmit register is empty
20	UCHANTXUNDER	AUDIO	U channel transmit register underrun
19	UCHANTXNEXTFIRST	AUDIO	U channel transmit register next byte will be first
18	IEC958-1 U/Q BUFFER ATTENTION	AUDIO	IEC 958 -1 U/Q channel buffer full interrupt

Table 9-14. Secondary Interrupt Sources (continued)

Interrupt	Interrupt Name	Module	Description
17	IEC958-2 CNEW	AUDIO	New C-channel received on IEC958-2
16	IEC958-2 VALNOGOOD	AUDIO	Validity flag not good on IEC958-2
15	IEC958-2 PARITY ERROR OR SYMBOL ERROR	AUDIO	IEC958-2 receiver parity error or symbol error
14	IEC958-2 U/Q BUFFER ATTENTION	AUDIO	IEC958-2 U/Q channel buffer full interrupt
13	U1CHANRCVOVER Q1CHANOVERRUN UQ1CHANERR	AUDIO	IEC958 receiver 1U/Q channel error
12	PDIR1UNOV	AUDIO	Processor data in 1 under / over
11	PDIR1RESYN	AUDIO	Processor data in 1 resync
10	PDIR2UNOV	AUDIO	Processor data in 2 under / over
9	PDIR2RESYN	AUDIO	Processor data in 2 resync
8	AUDIOTICK	AUDIO	“Tick” interrupt
7	U2CHANRCVOVER Q2CHANOVERRUN UQ2CHANERR	AUDIO	IEC 958 receiver 2 U/Q channel error
6	PDIR3 RESYNC	AUDIO	Processor data in 3 resync
5	PDIR3 FULL	AUDIO	Processor data in 3 full
4	IIS1TXEMPTY	AUDIO	I^2S1 transmit FIFO empty
3	IIS2TXEMPTY	AUDIO	I^2S2 transmit FIFO empty
2	EBUTXEMPTY	AUDIO	EBU transmit FIFO empty
1	PDIR2 FULL	AUDIO	Processor data in 2 full
0	PDIR1 FULL	AUDIO	Processor data in 1 full

Table 9-15. FlashMedia Interrupt Interface

FlashMediaIntStat FlashMediaIntEn FlashMediaIntClear bits	Int Name	Meaning	Reset Interrupt	Associated Interrupt
0	SHIFTBUSY1FALL	Interrupt set on falling edge of shift_busy_1	intClear	60
1	SHIFTBUSY1RISE	Interrupt set on rising edge of shift_busy_1	intClear	60
2	INTLEVEL1FALL	Interrupt set on falling edge of int_level_1	intClear	60
3	INTLEVEL1RISE	Interrupt set on rising edge of int_level_1	intClear	60
4	SHIFTBUSY2FALL	Interrupt set on falling edge of shift_busy_2	intClear	59
5	SHIFTBUSY2RISE	Interrupt set on rising edge of shift_busy_2	intClear	59
6	INTLEVEL2FALL	Interrupt set on falling edge of int_level_2	intClear	59
7	INTLEVEL2RISE	Interrupt set on rising edge of int_level_2	intClear	59

Table 9-15. FlashMedia Interrupt Interface (continued)

FlashMediaIntStat FlashMediaIntEn FlashMediaIntClear bits	Int Name	Meaning	Reset Interrupt	Associated Interrupt
8	RCV1FULL	Interrupt set if receive buffer reg 1 full	Read data	58
9	TX1EMPTY	Interrupt set if transmit buffer reg 1 empty	Write data	58
10	RCV2FULL	Interrupt set if receive buffer reg 2 full	Read data	57
11	TX2EMPTY	Interrupt set if transmit buffer reg 2 empty	Write data	57

9.4.3 Software Interrupts

The MCF5251 supports four software interrupts. These interrupts are activated by writing a 1 to an ExtraInt register bit. When active, the interrupts can generate a normal interrupt exception to the ColdFire processor. The interrupt exception is only generated if the corresponding level register interrupt mask is higher than the current processor interrupt mask.

9.4.4 Interrupt Monitor

To allow measuring interrupt latency during device debugging, two interrupt monitor pins INTMON1 and INTMON2 are available. These pins can be programmed in the Extraint register to output on any of the 64 interrupts available on the secondary interrupt controller.

Table 9-16. Extraint Register Descriptions

ExtraInt MBAR2 + 198 Bit Field	Name	Access	Description	Int No	Note
27:22	INTMON2	RW	INTMON2 selector		1, 4
21:16	INTMON1	RW	INTMON1 selector		1, 4
3, 7	SOFTINT3	R	Read softint3 value	50	1, 2
2, 6	SOFTINT2	R	Read softint2 value	49	1, 2
1, 5	SOFTINT1	R	Read softint1 value	48	1, 2
0, 4	SOFTINT0	R	Read softint0 value	47	1, 2
7	SOFTINT3_SET	W	Write one to this bit to set softint3	50	2, 3, 4
6	SOFTINT2_SET	W	Write one to this bit to set softint2	49	2, 3, 4
5	SOFTINT1_SET	W	Write one to this bit to set softint1	48	2, 3, 4
4	SOFTINT0_SET	W	Write one to this bit to set softint0	47	2, 3, 4
3	SOFTINT3_CLR	W	Write one to this bit to clear softint3	50	2, 3, 4
2	SOFTINT2_CLR	W	Write one to this bit to clear softint2	49	2, 3, 4
1	SOFTINT1_CLR	W	Write one to this bit to clear softint1	48	2, 3, 4
0	SOFTINT0_CLR	W	Write one to this bit to clear softint0	47	2, 3, 4

Notes:
1. INTMON1, INTMON2 registers are used to route an interrupt from the secondary interrupt controller (Section 9.3, “SIM Module Programming Registers”) such that the interrupt status can be monitored on either the external INTMON1 or INTMON2 pin. This feature is intended to help measure the latency of an interrupt service routine.
2. Bits 7–4 of the register can be read to obtain the value of the software interrupts 0–3. When written zero, the value of the corresponding software interrupt will not change. When written one, the corresponding software interrupt is set to a 1.
3. Bits 3–0 of the register can be read to read the value of software interrupts 0–3. When written zero, the value of the corresponding software interrupt will not change. When written one, the corresponding software interrupt is set to 0.
4. To write SOFTINT_SET and SOFTINT_CLR registers, while avoiding writing to the INTMONx fields, use word-size or byte-size addressing to update only bits 0–7.

9.5 System Protection and Reset Status Registers

This section provides the system Reset Status register and information about the Software Watchdog Timer and associated registers and descriptions.

9.5.1 Reset Status Register

The RSR contains a bit for each reset source to the SIM. A bit set to 1 indicates the last type of reset that occurred. The RSR is updated by the reset control logic on completion of the reset operation. Only one bit will be set at any given time in the RSR. If a reset occurs and the user failed to clear this register, reset control logic will clear all bits and set the appropriate bit to indicate the current cause of reset. The RSR programming model is illustrated as follows.

The Reset Status Register (RSR) is an 8-bit supervisor read-write register.

Address MBAR + $00 Access: User read/write

	7	6	5	4	3	2	1	0
R	HRST		SWTR					
W								
Reset	1	0	0	0	0	0	0	0

Figure 9-9. Reset Status Register (RSR)

Table 9-17. Reset Status Register (RSR) Field Descriptions

Field	Description
HRST	For the Hardware or System Reset, a 1 = An external device driving $\overline{\text{RSTI}}$ caused the last reset. Assertion of reset by an external device causes the core processor to take a reset exception. All registers in internal peripherals and the SIM are reset.
SWTR	For the Software Watchdog Timer Reset, a 1 = The last reset was caused by the software watchdog timer. If SWRI in the SYPCR is set and the software watchdog timer times out, a hardware reset occurs.

9.5.2 Software Watchdog Timer

The Software Watchdog Timer (SWT) prevents system lockup if the software become trapped in loops with no controlled exit.

The SWT can be enabled or disabled using the SWE bit in the SYPCR. If enabled, the SWT requires the execution of a software watchdog servicing sequence periodically. If this periodic servicing action does not occur, the SWT times-out resulting in a $\overline{\text{SWT IRQ}}$ or hardware reset, as programmed by the SWRI bit in the SYPCR.

If the SWT times out and the software watchdog transfer acknowledge enable (SWTA = SYPCR[2]) bit is set in the system protection control register, the $\overline{\text{SWT IRQ}}$ will assert. If after another timeout and the SWT IACK cycle has not occurred, the $\overline{\text{SWT TA}}$ signal will assert in an attempt to terminate the bus cycle and allow IACK cycle to proceed. The setting of the SWTAVAL flag bit (SYPCR[1]) in the system protection control register indicates that the $\overline{\text{SWT TA}}$ signal was asserted. The SWTA function when terminating a locked bus is shown in Figure 9-10.

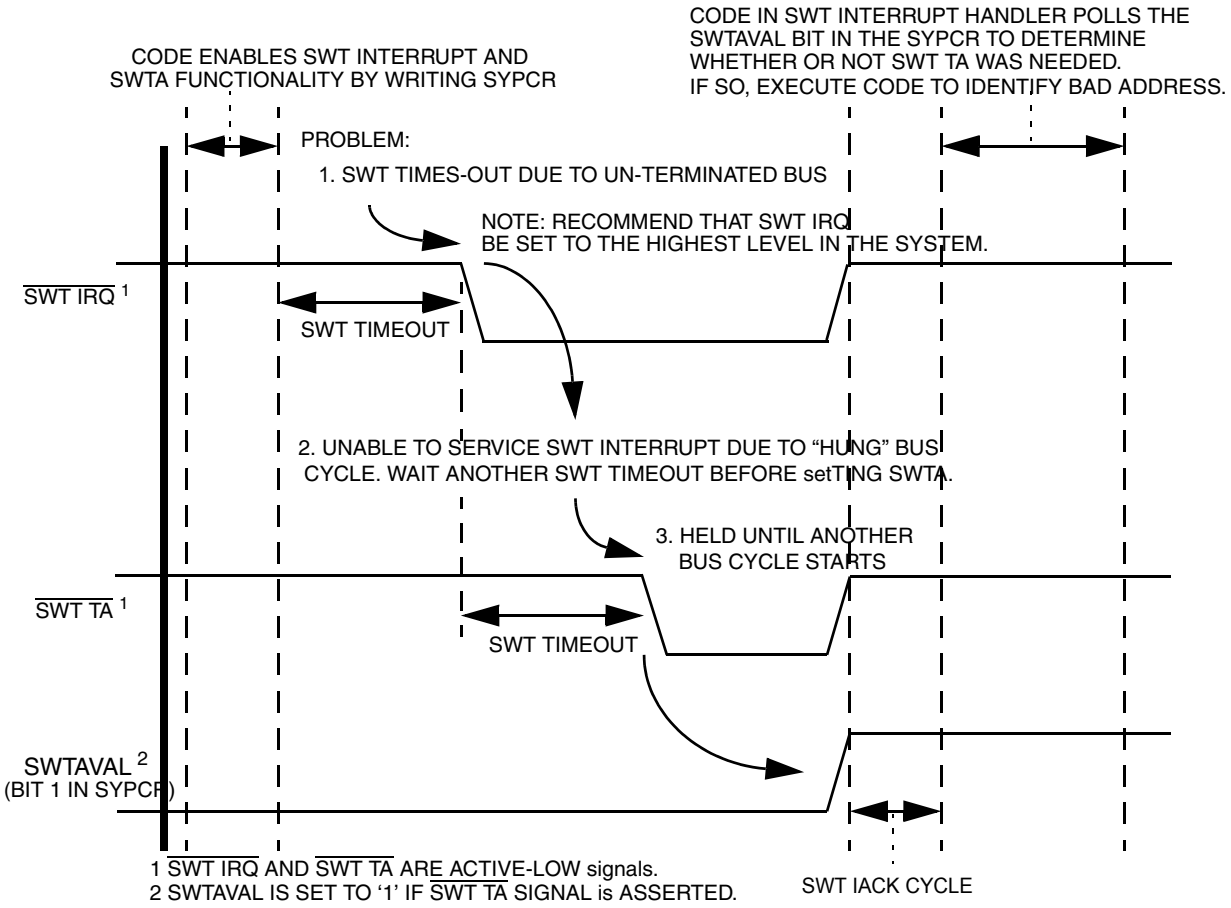


Figure 9-10. MCF5251 Unterminated Access Recovery

When the SWT times out and SWRI register bit is programmed for a software reset, an internal reset will be asserted, and the SWTR register bit will be set in the RSR.

To prevent SWT from interrupting or resetting, users must service the SWSR register. The SWT service sequence consists of the following steps:

1. Write $55 to SWSR
2. Write $AA to the SWSR

Both writes must occur in the order listed prior to the SWT timeout, but any number of instructions or accesses to the SWSR can be executed between the two writes. This allows interrupts and exceptions to occur, if necessary, between the two writes.

Caution should be exercised when changing system protection control register (SYPCR) values after the software watchdog timer (SWT) has been enabled with the setting of the SWE register bit, because it is difficult to determine the state of the SWT while the timer is running. The SWP and SWT[1:0] bits in SYPCR determine the SWT timeout period. The countdown value determined by the SWP and SWT[1:0] bits is constantly compared with that specified by these bits. Therefore, altering the contents of the SWP and SWT[1:0] bits improperly will result in unpredictable processor behavior. The following steps must be taken in order to change one of these values in the SYPCR:

1. Disable SWT by writing a 0 to the SWE bit in SYPCR.
2. Service the SWSR, write $55, then write $AA to SWSR. This action resets the counter.
3. Re-write new SWT[1:0] and SWP values to SYPCR register.
4. Re-enable SWT by writing a 1 to SWE bit in SYPCR. Users can perform this task in Step 3.

9.5.2.1 System Protection Control Register

The SYPCR controls the software watchdog timer, timeout periods, and software watchdog timer transfer acknowledge.

The SYPCR is an 8-bit read-write register. The register can be read at any time, but can be written only if $\overline{\text{SWT IRQ}}$ is not pending. At system reset, the software watchdog timer is disabled.

Address MBAR + $01 Access: User read/write

	7	6	5	4	3	2	1	0
R	SWE	SWRI	SWP	SWT[1]	SWT[0]	SWTA	SWTAVAL	
W								
Reset	0	0	0	0	0	0	0	–

Figure 9-11. System Protection Control Register (SYPCR)

Table 9-18. System Protection Control Register (SYPCR) Field Descriptions

Field	Description
7 SWE	Software Watchdog Enable 0 SWT disabled. 1 SWT enabled.
6 SWRI	Software Watchdog Reset/Interrupt Select 0 If SWT timeout occurs, SWT generates an interrupt to the core processor at the level programmed into the IL bits of ICR0. 1 SWT causes soft reset to be asserted for all modules of the part.
5 SWP	Software Watchdog Prescalar 0 SWT clock not prescaled. 1 SWT clock prescaled by a value of 8192.

Table 9-18. System Protection Control Register (SYPCR) Field Descriptions (continued)

Field	Description
4–3 SWT	The Software Watchdog Timing Delay bits (along with the SWP bit) select the timeout period for the SWT as shown in Table 9-19. At system reset, the software watchdog timer is set to the minimum timeout period.
2 SWTA	Software Watchdog Transfer Acknowledge Enable 0 SWTA Transfer Acknowledge disabled. 1 SWTA Assert Transfer Acknowledge enabled. After 1 SWT timeout period of the unacknowledged assertion of the SWT interrupt, the Software Watchdog Transfer Acknowledge will assert, which allows SWT to terminate a bus cycle and allow the IACK to occur.
1 SWTAVAL	Software Watchdog Transfer Acknowledge Valid 0 SWTA Transfer Acknowledge has NOT occurred. 1 SWTA Transfer Acknowledge has occurred. Write a 1 to clear this flag bit.
0	Reserved, should be cleared.

Table 9-19. SWT Timeout Period

SWP	SWT[1:0]	SWT TIMEOUT PERIOD
0	00	2^9 / BCLK
0	01	2^{11} / BCLK
0	10	2^{13} / BCLK
0	11	2^{15} / BCLK
1	00	2^{22} / BCLK
1	01	2^{24} / BCLK
1	10	2^{26} / BCLK
1	11	2^{28}/ BCLK

NOTE

If the SWP and SWT bits are modified to select a new software timeout, users must peform the software service sequence ($55 followed by $AA written to the SWSR) before the new timeout period takes effect.

9.5.2.2 Software Watchdog Interrupt Vector Register

The SWIVR contains the 8-bit interrupt vector the SIM returns during an interrupt- acknowledge cycle in response to a SWT-generated interrupt. The following register illustrates the SWIVR programming model.

The SWIVR is an 8-bit supervisor write-only register. This register is set to the uninitialized vector $0F at system reset.

Address MBAR + $02 Access: User read/write

	7	6	5	4	3	2	1	0
R W	SWIV7	SWIV6	SWIV5	SWIV4	SWIV3	SWIV2	SWIV1	SWIV0
Reset	0	0	0	0	1	1	1	1

Figure 9-12. Software Watchdog Interrupt Vector Register (SWIVR)

9.5.2.3 Software Watchdog Service Register

The SWSR is where the SWT servicing sequence should be written. To prevent an SWT timeout, users should write a $55 followed by a $AA to this register. Both writes must be performed in the order listed prior to the SWT timeout, but any number of instructions or accesses to the SWSR can be executed between the two writes. If the SWT has already timed out, writing to this register will have no effect in negating the SWT interrupt. The following register illustrates the SWSR programming model.

The SWSR is an 8-bit write-only register. At system reset, the contents of SWSR are uninitialized.

Address MBAR + $03 Access: User read/write

	7	6	5	4	3	2	1	0
R W	SWSR7	SWSR6	SWSR5	SWSR4	SWSR3	SWSR2	SWSR1	SWSR0
Reset	–	–	–	–	–	–	–	–

Figure 9-13. Software Watchdog Service Register (SWSR)

9.6 CPU HALT Instruction

Executing the CPU HALT instruction stops the core but does not disable any system clocks.

9.7 MCF5251 Bus Arbitration Control Registers

This section contains the Default Bus Master Park register, the internal arbitration operation, and the configuration of the PARK register bit.

9.7.1 Default Bus Master Park Register

The MPARK register determines the default bus master arbitration applied between internal transfers. This arbitration is needed because there are two bus masters inside the MCF5251. One is the CPU, the other is the DMA unit. Both can access internal registers within the MCF5251 peripherals. Table 9-14 shows the MPARK register bit encoding.

The MPARK is an 8-bit read-write register.

Address MBAR + $0c Access: User read/write

	7	6	5	4	3	2	1	0
R	PARK[1]	PARK[0]	IARBCTRL	EARBCTRL	SHOWDATA			BCR24BIT
W								
Reset	0	0	0	0	0	0	0	0

Figure 9-14. Default Bus Master Register (MPARK)

9.7.1.1 Internal Arbitration Operation

PARK register field bits [1:0] are programmed to indicate the priority of internal transfers. The possible masters that can initiate internal transfers are the core and the on-chip DMAs. Since the priority between DMAs is resolved by their relative priority amongst each other and by programming the BWC bits in their respective DMA control registers (see Section 14.4, "DMA Memory Map and Register Definitions"), the MPARK bits need only arbitrate priority between the core and the DMA module (which contains all four DMA channels) for internally generated transfers.

There are four arbitration schemes that the MPARK[1:0] bits can be programmed to with respect to internally generated transfers. The following summarizes these schemes when EARBCTRL=0:

1. Round Robin Scheme (PARK[1:0]=00)—In this scenario, depending on which master has priority in the current transfer, the other master has priority in the next transfer once the current master has finished. When the processor is initialized, the core has first priority.

   So for example, if the core is the bus master and is finishing a bus transfer and DMA channels 0 and 1 (both set to BWC=010) are asserting an internal bus request signal, then the DMA channel 0 would gain ownership of the bus after the core; but after channel 0 finishes its transfer, the core would have ownership of the bus if its request was asserted.

   NOTE

   The Internal DMA has higher priority than the ColdFire Core if the internal DMA has its bandwidth BWC[2:0] bits set to 000 (maximum bandwidth).

2. Park on Master Core Priority (PARK[1:0]=01)—Any time arbitration is occurring or the bus is idle, the core has priority. The DMA module can arbitrate a transfer only when the core's internal bus request signal is negated.
3. Park on Master DMA Priority (PARK[1:0]=10)—Any time arbitration is occurring or the bus is idle, the DMA has priority. The core can arbitrate a transfer only when the DMA's internal bus request signal is negated.
4. Park on Current Master Priority (PARK[1:0]=11)—Whatever the current master is, they have priority. Only when the bus is idle can the other master gain ownership and priority of the bus. For example, if out of reset the core has priority it will continue to have priority until the bus becomes idle. Then when the DMA asserts its internal bus request signal, it will then have priority.

9.7.1.2 PARK Register Bit Configuration

The following tables show the encoding for the PARK[1:0] bit of the MPARK register along with the priority schemes for each encoding.

Table 9-20. Default Bus Master Selected with PARK[1:0]

Park[1:0]	Default Bus Master Number
00	Round Robin between DMA and ColdFire Core
01	Park on master ColdFire Core
10	Park on master DMA Module
11	Park on current master

Table 9-21. Round Robin (PARK[1:0] = 00)

Current Highest Priority Master	Current Lowest Priority Master	Next Arbitration Cycle Highest Priority Master	Next Arbitration Cycle Lowest Priority Master
Core	DMA	DMA	Core
DMA	Core	Core	DMA

Table 9-22 shows the round robin configuration of internal module arbitration. Depending on which master has current ownership of the bus (i.e. has highest priority), the next arbitration cycle will switch priority to master that had lowest priority on that prior current cycle.

Table 9-22. Park on Master Core Priority (PARK[1:0] = 01)

Priority	Bus Master Name
Highest	ColdFire Core
Lowest	Internal DMA

Table 9-23. Park on Master DMA Module Priority (PARK[1:0] = 10)

Priority	Bus Master Name
Highest	Internal DMA
Lowest	ColdFire Core

Table 9-24. Park on Current Master Priority (PARK[1:0] = 11)

Current Highest Priority Master	Current Lowest Priority Master	Next Arbitration Cycle Highest Priority Master	Next Arbitration Cycle Lowest Priority Master
Core	DMA	Core	DMA
DMA	Core	DMA	Core

NOTE

When using the park on current master setting, the first master to arbitrate for the bus becomes the current master. The corresponding priority scheme should be interpreted as the priority of the next master once the current master finishes.

Table 9-25. Park Bit Descriptions

Bit Name	Description
IARBCTRL	Legacy bit. 0 Normal use 1 do not use
EARBCTRL	Legacy bit. 0 Normal use 1 do not use
SHOWDATA	Not used
BCR24BIT	This bit controls the BCR and address mapping for the DMA. The bit allows the byte count register to be used as a 24-bit register. See Section 14.4, "DMA Memory Map and Register Definitions" for memory maps and bit positions for the BCRs. 0 DMA BCRs function as 16-bit counters. 1 DMA BCRs function as 24-bit counters.

9.8 General Purpose I/Os

The MCF5251 has up to 57 programmable general-purpose outputs and up to 60 programmable general-purpose inputs. Two groups of 32-bit registers control these GPIOs.

Table 9-26. General Purpose I/O

Address	Name	Width	Description	Reset Value	Access
MBAR2 + $0x000	GPIO-READ	32	gpio input value		R
MBAR2 + $0x004	GPIO-OUT	32	gpio output value	0	R/W
MBAR2 + $0x008	GPIO-EN	32	output enable	0	R/W
MBAR2 + $0x00C	GPIO-FUNCTION	32	function select	0	R/W
MBAR2 + $0x0B0	GPIO1-READ	32	gpio input value	–	R
MBAR2 + $0x0B4	GPIO1-OUT	32	gpio output value	0	R/W
MBAR2 + $0x0B8	GPIO1-EN	32	output enable	0	R/W
MBAR2 + $0x0BC	GPIO1-FUNCTION	32	function select	0	R/W
MBAR2 + $0x0C0	GPIO-INT-STAT	32	interrupt status	–	R
MBAR2 + $0x0C0	GPIO-INT-CLEAR	32	interrupt clear	–	W
MBAR2 + $0x0C4	GPIO-INT-EN	32	interrupt enable	0	R/W

9.8.1 General Purpose Inputs

There are 60 possible general purpose inputs. They can be read in registers GPIO-READ and GPIO-READ1. These bits reflect the logical value of the pin they are associated with. The GPIO-READ and GPIO-READ1 registers always reflect the pin values, independent of the settings in the GPIO-FUNCTION, GPIO-EN, GPIO1-FUNCTION and GPIO1-EN registers. It does not matter if the pin is driving data out, or is being driven. The GPIO-READ and GPIO-READ1 bit to pin association is detailed in Table 9-27.

Table 9-27. General Purpose Input to Pin Mapping

General Purpose Input	Read From Pin
GPIO-READ(31)	$\overline{\text{IDE\_DIOR}}$/GPIO31
GPIO-READ(30)	$\overline{\text{BUFENB2}}$/GPIO30
GPIO-READ(29)	$\overline{\text{BUFENB1}}$/GPIO29
GPIO-READ(28)	$\overline{\text{CS1}}$/QSPICS3/GPIO28
GPIO-READ(27)	QSPIDOUT/SFSY/GPIO27
GPIO-READ(26)	RCK/QSPIDIN/QSPIDOUT/GPIO26
GPIO-READ(25)	QSPICLK/SUBR/GPIO25
GPIO-READ(24)	QSPICS2/MCLK2/GPIO24
GPIO-READ(23)	LRCK2/GPIO23
GPIO-READ(22)	SCLK2/GPIO22
GPIO-READ(21)	$\overline{\text{WAKEUP}}$/GPIO21
GPIO-READ(20)	SCLK1/GPIO20
GPIO-READ(19)	LRCK1/GPIO19
GPIO-READ(18)	SDATAO1/TOUT0/GPIO18
GPIO-READ(17)	SDATAI1/GPIO17
GPIO-READ(16)	QSPICS1/EBUOUT2/GPIO16
GPIO-READ(15)	QSPICS0/EBUIN4/GPIO15
GPIO-READ(14)	EBUIN3/CMD_SDIO2/GPIO14
GPIO-READ(13)	EBUIN2/SCLKOUT/GPIO13
GPIO-READ(12)	$\overline{\text{TA}}$/GPIO12
GPIO-READ(11)	MCLK1/GPIO11
GPIO-READ(10)	SCL1/TXD1/GPIO10
GPIO-READ(9)	none
GPIO-READ(8)	SDATAI3/GPIO8
GPIO-READ(7)	none
GPIO-READ(6)	EF/RXD2/GPIO6

General Purpose Input	Read From Pin
GPIO1-READ(63)	BCLKE/GPIO63
GPIO1-READ(62)	none
GPIO1-READ(61)	none
GPIO1-READ(60)	$\overline{\text{SDCS0}}$/GPIO60
GPIO1-READ(59)	$\overline{\text{SDRAS}}$/GPIO59
GPIO1-READ(58)	ADOUT/SCLK4/GPIO58
GPIO1-READ(57)	ADIN5/GPI57
GPIO1-READ(56)	ADIN4/GPI56
GPIO1-READ(55)	ADIN3/GPI55
GPIO1-READ(54)	ADIN2/GPI54
GPIO1-READ(53)	ADIN1/GPI53
GPIO1-READ(52)	ADIN0/GPI52
GPIO1-READ(51)	PSTCLK/GPIO51
GPIO1-READ(50)	PST0/GPIO50
GPIO1-READ(49)	PST1/GPIO49
GPIO1-READ(48)	PST2/INTMON2/GPIO48
GPIO1-READ(47)	PST3/INTMON1/GPIO47
GPIO1-READ(46)	RXD0/GPIO46
GPIO1-READ(45)	TXD0/GPIO45
GPIO1-READ(44)	SDA1/RXD1/GPIO44
GPIO1-READ(43)	LRCK3/AUDIOCLK/GPIO43
GPIO1-READ(42)	SDA0/SDATA3/GPIO42
GPIO1-READ(41)	SCL0/SDATA1_BS1/GPIO41
GPIO1-READ(40)	BCLK/GPIO40
GPIO1-READ(39)	$\overline{\text{SDCAS}}$/GPIO39
GPIO1-READ(38)	$\overline{\text{SDWE}}$/GPIO38

Table 9-27. General Purpose Input to Pin Mapping (continued)

General Purpose Input	Read From Pin
GPIO-READ(5)	CFLG/GPIO5
GPIO-READ(4)	DDATA3/$\overline{\text{RTS0}}$/GPIO4
GPIO-READ(3)	DDATA2/$\overline{\text{CTSO}}$/GPIO3
GPIO-READ(2)	DDATA1/$\overline{\text{RTS1}}$/SDATA2_BS2/GPIO2
GPIO-READ(1)	DDATA0/$\overline{\text{CTS1}}$/SDATA0_SDIO1/GPIO1
GPIO-READ(0)	XTRIM/TXD2/GPIO0

General Purpose Input	Read From Pin
GPIO1-READ(37)	EBUOUT1/GPIO37
GPIO1-READ(36)	EBUIN1/GPIO36
GPIO1-READ(35)	SCLK3/GPIO35
GPIO1-READ(34)	SDATAO2/GPIO34
GPIO1-READ(33)	$\overline{\text{IDE\_IORDY}}$/GPIO33
GPIO1-READ(32)	$\overline{\text{IDE\_DIOW}}$/GPIO32

NOTE

MCLK1 will output a clock signal just after reset and before it can be configured as a GPIO if so desired. The frequency of the clock will be the same as CRIN prior to initialization of the PLL.

NOTE

EBUOUT1 will output a clock signal just after reset and before they can be configured as GPIO. The frequency of the clock output will be CRIN/16.

These two pins can still be used for GPIO. The user needs to ensure that when one of these two pins is assigned as a GPIO control within the system, that its use will not cause the application to exhibit problems when the clock is active just after reset and before the boot code sets them to GPIO mode, e.g., do not use these pins to switch a critical circuit on/off.

9.8.1.1 General Purpose Input Interrupts

There are seven general purpose inputs, those associated with GPIO-READ(6:0), have interrupt capability. On every low-to-high edge transition of these inputs, one of the bits 0–6 of register GPIO-INT-STAT is set. On every high-to-low edge of the inputs, one of the bits 8–14 is set. Write 1 to clear to the corresponding bit in GPIO-INT-CLEAR register. If any bit in GPIO-INT-STAT is set, and the corresponding bit in GPIO-INT-EN is set, an interrupt will be made pending on the secondary interrupt controller.

NOTE

The registers GPIO-INT-STAT, GPIO-INT-CLEAR and GPIO-INT-EN also control some audio interrupts.

Set the GPIO_FUNCTION register bit to 1 or 0 for interrupts, as applicable.

Table 9-28. GPIO-INT-STAT, GPIO-INT-CLEAR and GPIO-INT-EN Interrupts

Event	GPIO-INT-STAT, GPIO-INT-CLEAR, GPIO-INT-EN Bit Number	Secondary Interrupt Controller Number
GPI0 L–H	0	32
GPI1 L–H	1	33
GPI2 L–H	2	34
GPI3 L–H	3	35
GPI4 L–H	4	36
GPI5 L–H	5	37
GPI6 L–H	6	38
GPI7 L–H	N/A	39
GPI0 H–L	8	32
GPI1 H–L	9	33
GPI2 H–L	10	34
GPI3 H–L	11	35
GPI4 H–L	12	36
GPI5 H–L	13	37
GPI6 H–L	14	38
GPI7 H–L	N/A	39
CD-ROM DECODER NEWBLOCK	16	56
CD-ROM DECODER ILSYNC	17	55
CD-ROM DECODER NOSYNC	18	54
CD-ROM DECODER CRCERROR	19	53
CD-ROM ENCODER NEWBLOCK	20	56
CD-ROM ENCODER ILSYNC	21	55
CD-ROM ENCODER NOSYNC	22	54
reserved	23	-

9.8.2 General Purpose Outputs

There are 57 possible general purpose outputs. They are controlled by registers GPIO-OUT, GPIO-EN, GPIO-FUNCTION, GPIO1-OUT, GPIO1-EN and GPIO1-FUNCTION. Three bits are needed to control a single general-purpose output. As an example, the logic that drives pin SCLK3/GPIO35 is shown in Figure 9-15.

The primary output function of the pin is the SCLK3 function it can be configured as a general-purpose output (GPIO35) by setting its controlling bit (35) in the GPIO1-FUNCTION register.

At power-on, the function is always the primary function. When a ‘0’ is programmed in any bit of GPIO-FUNCTION or GPIO1-FUNCTION, the corresponding pin gets its primary function. In this case, output drive strength and output value are determined by the primary function logic. When a ‘1’ is programmed the corresponding pin is in GPO-mode, drive direction is determined by value in GPIO-EN or GPIO1-EN. When a ‘0’ is programmed in any bit, the corresponding pin is driven to high-impedance state. When a ‘1’ is programmed, the corresponding pin is driven low or high.

When a pin is in GPO-mode, and being driven low-impedance, the actual drive value of the pin is determined by what is programmed in the corresponding bit of registers GPIO-OUT or GPIO1-OUT. If ‘0’ is programmed here, the pin is driven low. If ‘1’ is programmed, the pin is driven high.

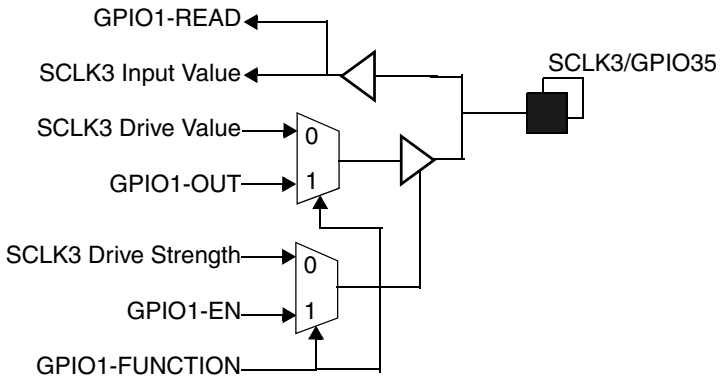


Figure 9-15. General-Purpose Pin Logic for Pin SCLK3/GPIO35

Table 9-29. General-Purpose Output Register Bits to Pins Mapping

GPIO-Function GPIO-EN GPIO-OUT Bit Number	Associated Pin	Pin Type	GPIO1-Function GPIO1-EN GPIO1-OUT Bit Number	Associated Pin	Pin Type
31	$\overline{\text{IDE\_DIOR}}$/GPIO31	I/O	63	BCLKE/GPIO63	I/O
30	$\overline{\text{BUFENB2}}$/GPIO30	I/O	62	none	I/O
29	$\overline{\text{BUFENB1}}$/GPIO29	I/O	61	none	I/O
28	$\overline{\text{CS1}}$/QSPICS3/GPIO28	I/O	60	$\overline{\text{SD\_CS0}}$/GPIO60	I/O
27	QSPIDOUT/SFSY/GPIO27	I/O	59	$\overline{\text{SDRAS}}$/GPIO59	I/O
26	RCK/QSPIDIN/QSPI DOUT/GPIO26	I/O	58	ADOUT/SCLK4/GPIO58	I/O
25	QSPICLK/SUBR/GPIO25	I/O	57	none	I/O
24	QSPICS2/MCLK2/GPIO24	I/O	56	none	I/O
23	LRCK2/GPIO23	I/O	55	none	I/O
22	SCLK2/GPIO22	I/O	54	A23/GPO54	O
21	$\overline{\text{WAKEUP}}$/GPIO21	I/O	53	$\overline{\text{SDUDQM}}$/GPO53	O
20	SCLK1/GPIO20	I/O	52	$\overline{\text{SDLDQM}}$/GPO52	O
19	LRCK1/GPIO19	I/O	51	PSTCLK/GPIO51	I/O
18	SDATAO1/TOUT0/GPIO18	I/O	50	PST0/GPIO50	I/O
17	SDATAI1/GPIO17	I/O	49	PST1/GPIO49	I/O

Table 9-29. General-Purpose Output Register Bits to Pins Mapping (continued)

GPIO-Function GPIO-EN GPIO-OUT Bit Number	Associated Pin	Pin Type	GPIO1-Function GPIO1-EN GPIO1-OUT Bit Number	Associated Pin	Pin Type
16	QSPICS1/EBUOUT2/GPIO16	I/O	48	PST2/INTMON2/GPIO48	I/O
15	QSPICS0/EBUIN4/GPIO15	I/O	47	PST3/INTMON1/GPIO47	I/O
14	EBUIN3/CMD_SDIO2/GPIO14	I/O	46	RXD0/GPIO46	I/O
13	EBUIN2/SCLKOUT/GPIO13	I/O	45	TXD0/GPIO45	I/O
12	$\overline{TA}$/GPIO12	I/O	44	SDA1/RXD1/GPIO44	I/O
11	MCLK1/GPIO11	I/O	43	LRCK3/AUDIOCLK/GPIO43	I/O
10	SCL1/TXD1/GPIO10	I/O	42	SDA0/SDATA3/GPIO42	I/O
9	none		41	SCL0/SDATA1_BS1/GPIO41	I/O
8	SDATAI3/GPIO8	I/O	40	BCLK/GPIO40	I/O
7	none	I/O	39	$\overline{SDCAS}$/GPIO39	I/O
6	EF/RXD2/GPIO6	I/O	38	$\overline{SDWE}$/GPIO38	I/O
5	CFLG/GPIO5	I/O	37	EBUOUT1/GPIO37	I/O
4	DDATA3/$\overline{RTS0}$/GPIO4	I/O	36	EBUIN1/GPIO36	I/O
3	DDATA2/$\overline{CTS0}$/GPIO3	I/O	35	SCLK3/GPIO35	I/O
2	DDATA1/$\overline{RTS1}$/SDATA2_ BS2/GPIO2	I/O	34	SDATAO2/GPIO34	I/O
1	DDATA0/$\overline{CTS1}$/SDATA0_ SDIO1/GPIO1	I/O	33	$\overline{IDE\_IORDY}$/GPIO33	I/O
0	XTRIM/TXD2/GPIO0	I/O	32	$\overline{IDE\_DIOW}$/GPIO32	I/O

9.9 Multiplexed Pin Configuration

The MCF5251 has a number of pins which are multiplexed with both a primary function, a secondary function, and a GPIO function (triple functionality). Two pins have 3 major functions (a primary and two secondary functions). Two pins also have their multiplexed functions selected at power-on reset via external pull-up / pull-down resistors.

At Power-on RESET the primary function (the function listed first in the pin name) is enabled. The GPIO function is selected as required by setting the appropriate bit in the GPIO-FUNCTION or GPIO1-FUNCTION registers as described in Section 9.8, “General Purpose I/Os”.
To enable the secondary function the appropriate Pin Configuration register bit needs to be set.

Note: in all cases the GPIO FUNCTION register setting has priority. Therefore it is necessary to set the appropriate GPIO FUNCTION bit to 0 to enable the primary or secondary function.

Address MBAR2 + $19c Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	A24	EBU IN4	EBU OUT2	MCLK2	SUBR	QSPI_DIN/ DOUT	QSPI_ CS3	RTS1	SDATA2 _BS2	CTS0	RTS0	TXD1	RXD1	INT MON1	INT MON2	sclk _out
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	cmd _sdio2	CTS1	SDATA0 _SDIO1	SDATA1 _BS1	SDATA3	SFSY	SCLK4	TOUT0							RXD2	TXD2
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-16. Pin Configuration Register

Table 9-30 lists the pins which have a triple multiplexed function and the associated Pin Configuration bit. (Pins configured at power-on reset by pull-up / pull-down resistors are listed for reference).

Table 9-30. Triple Multiplexed Pins

Pin	Bit	Name	Description
J3		$\overline{CS0}$/$\overline{CS4}$	Function select with pull-up/pull-down resistor connected to A23 pin.
M10		LRCK3/AUDIOCLK/GPIO43	Audioclock input selected by pull-up/pull-down resistor connected to A20/A24.
R6	0	XTRIM/TXD2/GPIO0	0 XTRIM 1 TXD2
R9	1	EF/RXD2/GPIO6	0 EF 1 RXD2
R8	8	SDATAO1/TOUT0/GPIO18	0 SDATAO1 1 TOUT0
J5	9	ADOUT/SCLK4/GPIO58	0 ADOUT 1 SCLK4
M8	10	QSPIDOUT/SFSY/GPIO27	0 QSPIDOUT 1 SFSY
K9	11	SDA0/SDATA3/GPIO42	0 SDA0 1 SDATA3
P10	12	SCL0/SDATA1_BS1/GPIO41	0 SCL0 1 SDATA1_BS1
K10	13 + 14	DDATA0/$\overline{CTS1}$/SDATA0_SDIO1/GPIO1	14–13 0: 0 DDATA0 0: 1 SDATA0SDIO1 1: 0 CTS1 1: 1 CTS1
K7	15	EBUIN3/CMD_SDIO2/GPIO14	0 EBUIN3 1 CMDSDIO2
M6	16	EBUIN2/SCLKOUT/GPIO13	0 EBUIN2 1 SCLKOUT

Table 9-30. Triple Multiplexed Pins (continued)

Pin	Bit	Name	Description
H14	17	PST2/INTMON2/GPIO48	0 PST2 1 INTMON2
H13	18	PST3/INTMON1/GPIO47	0 PST3 1 INTMON1
J15	19	SDA1/RXD1/GPIO44	0 SDA1 1 RXD0
J13	20	SCL1/TXD1/GPIO10	0 SCL1 1 TXD0
J12	21	DDATA3/$\overline{\text{RTS0}}$/GPIO4	0 DDATA3 1 RTS0
J14	22	DDATA2/$\overline{\text{CTS0}}$/GPIO3	0 DDATA2 1 CTS0
R11	23 + 24	DDATA1/$\overline{\text{RTS1}}$/SDATA2BS2/GPIO2	24: 23 0: 0 DDATA1 0: 1 SDATA2BS2 1: 0 RTS1 1: 1 RTS1
M7	25	$\overline{\text{CS1}}$/QSPICS3/GPIO28	0 CS1 1 QSPICS3
N7	26	RCK/QSPIDIN/QSPIDOUT/GPIO26	0 RCK 1 QSPIDIN / QSPIDOUT Note: QSPIDOUT is selected when CS3 is active, otherwise QSPIDIN is enabled.
P7	27	QSPICLK/SUBR/GPIO25	0 QSPICLK 1 SUBR
P9	28	QSPICS2/MCLK2/GPIO24	0 QSPICS2 1 MCLK2
N8	29	QSPICS1/EBUOUT2/GPIO16	0 QSPICS1 1 EBUOUT2
R7	30	QSPICS0/EBUIN4/GPIO15	0 QSPICS0 1 EBUIN4
F6	31	A20/A24	0 A20 1 A24

Chapter 10
Chip Select Module

The Chip Select Module provides user-programmable control of the three chip select outputs, two buffer enable outputs and one output-enable signal. This chapter describes the operation, memory map, and register descriptions of the Chip Select module, including the chip select address, mask, and control registers.

10.1 Chip Select Features

- Three programmable chip select outputs ($\overline{CS0}/\overline{CS4}$, $\overline{CS1}$ and $\overline{CS2}$ ($\overline{IDE\_DIOR}$ and $\overline{IDE\_DIOW}$))
- $\overline{IORDY}$ and $\overline{TA}$ handshake pins
- Two programmable buffers enable signals for glueless interface to bus buffers
- Address masking for memory block sizes from 64 Kbytes to 4Gbytes
- Programmable wait states
- Port size is 16 bits

10.2 Chip Select Signals

The MCF5251 provides three programmable chip selects that can directly interface with SRAM, EPROM, EEPROM, and peripherals. Chip select CS2 provides separate read and write strobes for an AT-bus peripheral interface, and uses $\overline{IORDY}$ signalling to insert wait states.

10.2.1 $\overline{CS0}/\overline{CS4}$

$\overline{CS0}$ is the first chip select and it addresses either the on-chip or external boot memory. At power-on reset, all bus cycles are mapped to the $\overline{CS0}$. This allows the boot memory to be defined at any address space. $\overline{CS0}$ is the only chip select initialized at reset.

During power-on reset, pin A23 is sensed. A resistor should be connected between this pin and VDD or GND. Depending whether a pull-up or pull-down is mounted, different options are selected as described in Table 10-1.

Table 10-1. CS0 Operation

Pin	Description
A23	Pull-up: Boot from memory connected to $\overline{CS0}/\overline{CS4}$. $\overline{CS0}/\overline{CS4}$ function is $\overline{CS0}$ Pull-down: Boot from on-chip boot ROM. $\overline{CS0}/\overline{CS4}$ function becomes $\overline{CS4}$

10.2.2 $\overline{CS1}$/QSPI_CS3/GPIO28

$\overline{CS1}$ has a programmable address range, wait state generation, and internal/external termination. A reset clears all chip select programming. It is multiplexed with QSPI_CS3 and GPIO28.

10.2.3 CS2 — $\overline{IDE\_DIOR}$/GPIO31 and $\overline{IDE\_DIOW}$/GPIO32

CS2 provides two separate control signals for read and write operations. These two signals go active during CS2 cycles. $\overline{IDE\_DIOR}$ can be programmed to go active on read and write cycles, or only to go active on read cycles. $\overline{IDE\_DIOW}$ operates only on write cycles.

$\overline{IDE\_DIOR}$ and $\overline{IDE\_DIOW}$ can be used as enables to access an IDE drive or another AT-bus peripheral. This added functionality allows users to insert more than 16 wait states on $\overline{IDE\_DIOR}$, $\overline{IDE\_DIOW}$, and allows dynamic cycle termination using the $\overline{IDE\_IORDY}$ signal.

10.2.4 CS3

There is no physical output pin. However, the registers for CS3 are available and can be used to enable the BUFENx outputs. These BUFENx outputs could then be used as a physical CS3. This would require programming the CS3 registers and then setting the appropriate bits in the IDECONFIG1 register. See Table 13-2, IDEConfig1 Register Field Descriptions for bit 18 or 21 settings.

10.2.5 Output Enable Signal $\overline{OE}$

The $\overline{OE}$ signal enables read accesses to memory and/or peripherals. It is asserted and negated on the falling edge of the clock. This signal is asserted when there is a match with one of the chip selects.

10.2.6 Buffer Enable – $\overline{BUFENB1}$ and $\overline{BUFENB2}$ Signals

The $\overline{BUFENB1}$/GPIO29 and $\overline{BUFENB2}$/GPIO30 signals are intended to enable bus buffers which will provide isolation / buffering between the MCF5251 high speed memory bus and additional external memory mapped devices.

$\overline{BUFENB1}$ is always active on $\overline{CS0}$.
$\overline{BUFENB2}$ is always inactive on $\overline{CS0}$. It is programmable to be active on $\overline{CS1}$, $\overline{CS2}$, $\overline{CS3}$ (special case) and $\overline{CS4}$ as desired.

10.2.7 Bus Termination Signal – $\overline{IDE\_IORDY}$

The $\overline{IDE\_IORDY}$ signal controls the insertion of wait states on $\overline{CS2}$.

10.3 Chip Select Operation

The chip select module provides a glueless interface to many types of external memory. The module contains the necessary external control signals to interface to SRAM, PROM, EPROM, EEPROM, FLASH and peripherals.

Some features of the chip selects are controlled by the IDECONFIG1 and IDECONFIG2 registers. These are described in Section 10.4, "Chip Select Memory Map and Register Definitions."

Each of the three chip select outputs has an associated mask register and control register.

Chip selects ($\overline{CS0}$/$\overline{CS4}$, $\overline{CS1}$/QSPI_CS3/GPIO28, $\overline{IDE\_DIOR}$ / $\overline{IDE\_DIOW}$ (CS2):

- Each has a 16-bit base address register.
- Each has a 32-bit mask register, which provides 16-bit address masking and access control.
- Each has a 16-bit control register, which provides port size, burst capability, wait state generation and automatic acknowledge generation.

$\overline{CS0}$ provides special functionality. It is a "global" chip select after reset and provides relocatable boot ROM capability.

In addition to the two external chip select outputs, the module contains one chip select ($\overline{CS2}$) for use with AT-bus peripherals such as IDE drives and Flash Card interfaces. Capabilities for $\overline{CS2}$ are like $\overline{CS1}$, but there are some enhancements for typical AT-bus features. The enhancements are described in Section 10.4, "Chip Select Memory Map and Register Definitions."

10.3.1 General-Purpose Chip Select Operation

The general-purpose chip selects are controlled by the chip select mask register (CSMR), the chip select control register (CSCR), and by the chip select address register (CSAR). There is one CSAR, CSMR, and CSCR for each of the chip selects ($\overline{CS0}$, $\overline{CS1}$, CS2 and $\overline{CS4}$).

Chip Selects:

- The chip select address register controls the base address space of the chip select.
- The chip select mask register controls the memory block size and addressing attributes of the chip select.
- The chip select control register programs the features of the chip select signals.

10.3.2 Port Sizing

The MCF5251 only supports a 16-bit wide port size (PS). The size of the port controlled by a chip-select is programmable. The port size is specified by the (PS) bits in the chip select control register (CSCR). It should always be programmed as a 16-bit wide port. See Section 10.4.2.3, "Chip Select Control Register," for details.

10.3.3 Global Chip-Select Operation

$\overline{CS0}$ is the global (boot) chip select and it allows address decoding for the boot ROM before system initialization occurs. Its operation differs from the other external chip-select outputs following a system reset. Its operation is also dependent on the pull-up or pull-down status of address line A23 at power-on reset, see Section 10.2.1, "CS0/CS4," for details.

NOTE

$\overline{CS0}$/$\overline{CS4}$ are multiplexed, when $\overline{CS0}$ is enabled for on-chip boot ROM access, $\overline{CS0}$ is used for these access and $\overline{CS4}$ is automatically enabled as the output for the $\overline{CS0}$/$\overline{CS4}$ pin.

After system reset, $\overline{CS0}$ is asserted for every external access. Internal accesses can be made to go external by setting the internal bus arbitration control (IARBCTRL) bit of the default bus master (MPARK) register in the system integration module (SIM). No other chip-select can be used while $\overline{CS0}$ is a global chip select. $\overline{CS0}$ operates in this manner until the valid bit is set in chip select mask register CSMR0[0], at which point $\overline{CS1}$ may be used. At reset, the port size and automatic acknowledge functions of the global chip-select are determined.

The reset state of $\overline{CS0}$ is always auto-acknowledge (AA) with 15 wait states and the port size is 16-bits.

Provided the required address range is first loaded into chip select address register (CSAR), $\overline{CS0}$ can be programmed to continue to decode for a range of addresses after the valid (V) bit is set. After the V-bit is set for $\overline{CS0}$, global chip-select can be restored only with another system reset.

10.4 Chip Select Memory Map and Register Definitions

The Chip Select module registers and their field descriptions are provided in this section followed by a code example to initialize the chip selects.

10.4.1 Chip Select Register Memory Map

Table 10-2 shows the memory map of all the chip-select registers. Reading reserved locations returns zeros.

The CSCRs should be accessed through a MOV.L to longword address offset they belong to, while reading and writing to the lower 16-bits of the longword data transfer (DATA[15:0]).

NOTE

All of these accesses are longword in length, instead of word length, even though both the CSARs and CSCRs use only 16 bits in the 32-bits registers.

Table 10-2. Memory Map of Chip-Select Registers

Address[1]	Name	Width	Description	Reset[2]	Access
MBAR + 0x80	CSAR0	16	Chip-Select Address Register–Bank 0	Uninitialized	R/W
MBAR + 0x84	CSMR0	32	Chip-Select Mask Register–Bank 0	Uninitialized (except V = 0)	R/W

Table 10-2. Memory Map of Chip-Select Registers (continued)

Address[1]	Name	Width	Description	Reset[2]	Access
MBAR + 0x88	CSCR0	16	Chip-Select Control Register–Bank 0	BEM = 1; BSTR = BSTW = 0; AA =; PS1 =; PS0 =; WS3 = WS2 = WS1 = WS0 = 1	R/W
MBAR + 0x8C	CSAR1	16	Chip-Select Address Register–Bank 1	Uninitialized	R/W
MBAR + 0x90	CSMR1	32	Chip-Select Mask Register–Bank 1	Uninitialized (except V = 0)	R/W
MBAR + 0x94	CSCR1	16	Chip-Select Control Register–Bank 1	Uninitialized	R/W
MBAR + 0x98	CSAR2	16	Chip-Select Address Register–IDE	Uninitialized	R/W
MBAR + 0x9C	CSMR2	32	Chip-Select Mask Register–IDE	Uninitialized (except V = 0)	R/W
MBAR + 0xA0	CSCR2	16	Chip-Select Control Register–IDE	Uninitialized	R/W
MBAR + 0xA4	CSAR3	16	Chip-Select Address Register	Uninitialized	R/W
MBAR + 0xA8	CSMR3	32	Chip-Select Mask Register	Uninitialized (except V = 0)	R/W
MBAR + 0xAC	CSCR3	16	Chip-Select Control Register	Uninitialized	R/W
MBAR + 0xB0	CSAR4	16	Chip-Select Address Register–Bank 4	Uninitialized	R/W
MBAR + 0xB4	CSMR4	32	Chip-Select Mask Register–Bank 4	Uninitialized (except V = 0)	R/W
MBAR + 0xB8	CSCR4	16	Chip-Select Control Register–Bank 4	Uninitialized	R/W

1 Addresses not assigned to a register and undefined register bits are reserved for future expansion. Write accesses to these reserved address spaces and reserved register bits are undefined.

2 The reset value column indicates the register initial value at reset.

10.4.2 Chip Select Module Registers

The various chip select registers in the module are described in this section.

10.4.2.1 Chip Select Address Register

The Chip Select Address registers (CSAR*x*) determine the base address of the corresponding chip select pin. These read/write registers are 32-bit in length.

Address MBAR + 0x80
MBAR + 0x8C
MBAR + 0x98
MBAR + 0xA4

Access: User read/write

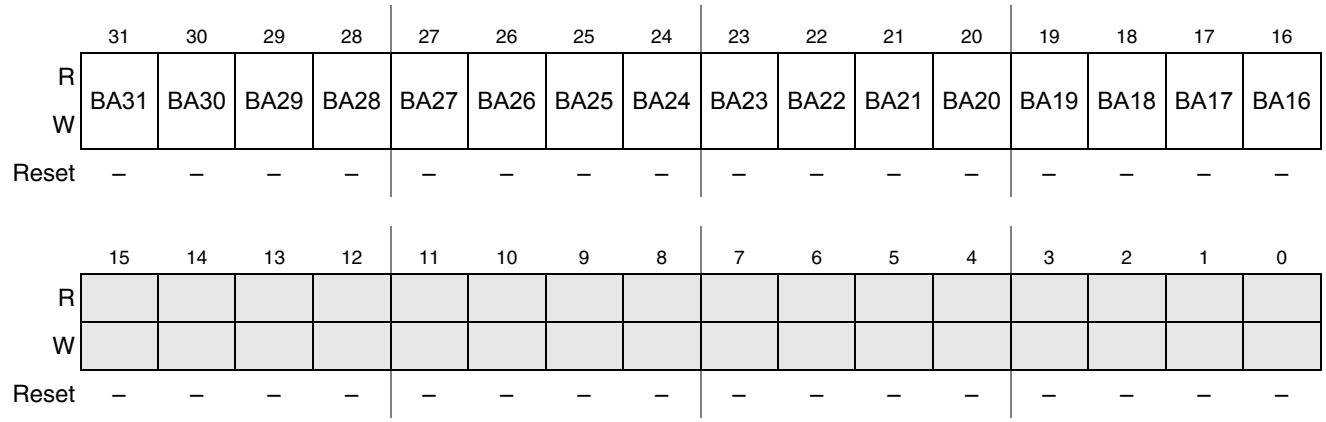


Figure 10-1. Chip Select Address Register (CSAR*x*)

Table 10-3. Chip Select Register (CSAR*x*) Field Descriptions

Field	Description
31–16 BA	The Base Address field defines the base address location of memory dedicated to chip select $\overline{CS}$[3:0]. These bits are compared to bits 31–16 on the internal core address bus to determine if the chip select memory is being accessed.
15–0	Reserved

10.4.2.2 Chip Select Mask Register

The chip select mask registers CSMRx determine the address mask. In addition, they determine what type of access is allowed for these signals. Each CSMR is a 32-bit read/write control register that physically resides in the chip select module. With the exception of bit 0 (V-bit), which is initialized to 0 on reset, all other bits in CSMRx are uninitialized by reset.

Address MBAR + 0x84
MBAR + 0x90
MBAR + 0x9C
MBAR + 0xA8

Access: User read/write

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W								WP		AM	C/I	SC	SD	UC	UD	V
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0

Figure 10-2. Chip Select Mask Register (CSMR*x*)

Table 10-4. Chip Select Mask Register (CSMR*x*) Field Descriptions

Field	Description
31–16 BAM	The Base Address Mask field defines the chip select block size through the use of address mask bits. Any set bit masks the corresponding base address register (CSAR) bit (the base address bit becomes 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 all $\overline{CS}$ are equal to 2^n, where n = (number of bits set in the base address mask field of the respective CSMR + 16). For example, if CSAR0 were set at $0000 and CSMR0 were set at $0008, then chip select $\overline{CS0}$ would address two discontinuous memory blocks of 64Kbytes each: the first block would be from $00000000 to $0000FFFF and the second block would be from $00080000 to $0008FFFF. Stated another way, if any of the upper 16-bits in the CSMR0 were set, then the corresponding address bit is a don't care in the chip select decode. Another example might be if $\overline{CS0}$ were to access 32MBs of address space starting at location $0 and $\overline{CS1}$ has to begin at the next byte after $\overline{CS0}$ for an address space of 16MB. Then: CSAR0 = $0000, (upper 16 bits of) CSMR0 = $01FF, and CSAR1 = $0200, (upper 16 bits of) CSMR1 = $00FF.
15–9, 7	Reserved
Address Space Mask Bits	
8 WP	The Write Protect bit can restrict write accesses to the address range in a CSAR. An attempt to write to the range of addresses specified in a CSAR that has this bit set results in the appropriate chip select not being selected. No exception occurs. 0 Both read and write accesses are allowed. 1 Only read access is allowed.

Table 10-4. Chip Select Mask Register (CSMR*x*) Field Descriptions (continued)

Field	Description
6–1 AM, C/I, SC, SD, UC, UD	These fields mask specific address spaces. If an address space mask bit were cleared, an access to a location in that address space can activate the corresponding chip select. If an address space mask bit were set, an access to a location in that address space becomes a regular external bus access, and no chip select is activated. AM: Alternate master access (DMA) C/I: Interrupt cycle access SC: Supervisor code access SD: Supervisor data access UC: User code access UD: User data access For each address space mask bit (AM, C/I, SC, SD, UC, UD): 0 Do not mask this address space for the chip select. An access using the chip select can occur for this address space. 1 Mask this address space from the chip select activation. If this address space is accessed, no chip select activation occurs on the external cycle.
0 V	The Valid bit indicates that the contents of its address register, mask register, and control register are valid. The programmed chip selects do not assert until the V-bit is set (except for $\overline{CS0}$ which acts as the global (boot) chip select—see Section 10.3.3, “Global Chip-Select Operation.”) A reset clears the V-bit in each CSMR. 0 Chip select invalid 1 Chip select valid

10.4.2.3 Chip Select Control Register

CSCRx control the auto acknowledge, external master support, port size, burst capability, and activation of each of the chip selects.

For CSCR0, bits BSTR, and BSTW are initialized to 0 by reset; bits WS[3:0] and BEM are initialized to 1 by reset; while AA, PS1, and PS0 are loaded with “110”, respectively at reset. For CSCR1 to CSCR4 none of the bits are initialized at reset. These are shown in Figure 10-3 and Figure 10-4.

$\overline{CS0}$ is the global (boot) chip select which allows address decoding for boot ROM before system initialization occurs. Its operation differs from the other external chip select outputs following a system reset.

Address MBAR + 0x8A — Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R			WS3	WS2	WS1	WS0		AA	PS1	PS0		BSTR	BSTW			
W																
Reset	–	–	1	1	1	1	–	1	1	0	–	0	0	–	–	0

Figure 10-3. Chip Select Control Register (CSCR0)

Address MBAR +0x96
MBAR +0xA2
MBAR +0xAE

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R			WS3	WS2	WS1	WS0		AA	PS1	PS0		BSTR	BSTW			
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0

Figure 10-4. Chip Select Control Registers (CSCR*x*)

Table 10-5. Chip Select Control Register (CSCR*x*) Field Descriptions

Field	Description
15–14	Reserved.
13–10 WS	The Wait States field defines the number of wait states that are inserted before an internal transfer acknowledge is generated. If the AA bit is cleared, $\overline{TA}$ must be asserted by the external system regardless of the number of wait states generated.
9	Reserved.
8 AA	The Auto-Acknowledge Enable field determines the assertion of the internal transfer-acknowledge for accesses specified by the chip select address. 0 No internal transfer acknowledge ($\overline{TA}$) is asserted. 1 Internal acknowledge ($\overline{TA}$) is asserted as specified by WS[3:0].
7–6 PS	The Port Size field specifies the width of the data associated with each chip select. It determines where data is driven during write cycles and where data is sampled during read cycles. Port size should always be programmed to 16-bits. 00 Reserved. 01 Reserved. 10 16-bit port size–Data sampled and driven on D[31:16] only. 11 16-bit port size–Data sampled and driven on D[31:16] only. **Note:** A0 is not available on the external bus.
5	Reserved.
4 BSTR	The Burst Read Enable field specifies whether burst reads are used for the memory associated with each chip select. 0 Breaks data larger than the specified port size into individual non-burst reads that equals the specified port size. For example, a longword read from an 16-bit port would be broken into two individual wordreads. 1 Enables burst read of data larger than the specified port size.
3 BSTW	The Burst Write Enable field specifies whether burst writes are used for the memory associated with each chip select. 0 Break data larger than the specified port size into individual non-burst writes that equals the specified port size. For example, a longword write to an 16-bit port would be broken into two individual word writes. 1 Enables burst write of data larger than the specified port size.
2–0	Reserved.

10.4.2.4 Code Example

The following code provides an example of how to initialize the chip-selects.

```
CSAR0 EQU MBARx+$080;Chip Select 0 address register
CSMR0 EQU MBARx+$084;Chip Select 0 mask register
CSCR0 EQU MBARx+$088;Chip Select 0 control register

CSAR1 EQU MBARx+$08C;Chip Select 1 address register
CSMR1 EQU MBARx+$090;Chip Select 1 mask register
CSCR1 EQU MBARx+$094;Chip Select 1 control register

        ; All other chip selects should be programmed and made valid before global
        ; chip select is de-activated by validating CS0
        ; Program Chip Select 1 Registers

move.l#$00000000,D0;CSAR1 base addresses $00000000 (to $001FFFFF)
move.lD0,CSAR1;and $80000000 (to $801FFFFF)

move.l#$000009B0,D0;CSCR1 = 2 wait states, AA=1, PS=16-bit, BEM=1,
move.lD0,CSCR1;BSTR=1, BSTW=0

move.l#801F0001,D0;Address range from $00000000 to $001FFFFF and
move.lD0,CSMR1;$80000000 to $801FFFFF
;WP,EM,C/I,SC,SD,UC,UD=0, V=1


;Program Chip Select 0 Registers
move.l#$00800000,D0;CSAR0 base address $00800000 (to $009FFFFF)
move.lD0,CSAR0

move.l#$00000D80,D0;CSCR0 = 3 wait states, AA=1, PS=16-bit, BEM=0,
move.lD0,CSCR0;BSTR=0, BSTW=0

; Program Chip Select 0 Mask Register (validate chip selects)
move.l#001F0001,D0;Address range from $00800000 to $009FFFFF
move.lD0,CSMR0;WP,EM,C/I,SC,SD,UC,UD=0; V=1
```

Chapter 11
General Purpose Timer Modules

This chapter describes the configuration and operation of the two general purpose timer modules (Timer0 and Timer1) in the MCF5251. The memory map, register descriptions, and example initialization code are also provided.

11.1 Timer Module Overview

The MCF5251 incorporates two independent, general-purpose 16-bit timers. The output of an 8-bit prescaler clocks each 16-bit timer. The prescaler input can be the system clock or the system clock divided by 16. Timer0 output pin is multiplexed with SDATAO1/TOUT0/GPIO18. Upon reset, this pin is programmed as SDATAO1. To use the TOUT0 pin function it is necessary to program the Pin Configuration register appropriately.

NOTE

The maximum system clock (SYSCLK) is 1/2 CPU clock.

11.2 Timer Features

Each of the general purpose 16-bit timers provide the following features:

- Maximum period of 3.83 seconds at 70 MHz SYSCLK (BCLK)
- 14.3 ns minimum resolution at 70 MHz
- Programmable sources for the clock input
- Output-compare with programmable mode for the output pin (Timer0 only)
- Free run and restart modes
- Maskable interrupt on reference-compare
- Programmable to count and compare to a reference value stored in a register
- An 8-bit prescalar output clocks the timers
- Users can program the prescalar clock input
- Programmed events generate interrupts
- Users can configure the TOUT0 pin to toggle or to pulse on an event

The minimum resolution of each timer is one system clock (SYSCLK) cycle (14.3 ns at 70 MHz). The maximum timeout period (16 × 256 × 65536) ÷ 70 MHz = 3.83 seconds. ($0 - $FFFF = 65536 decimal.)

11.3 Block Diagram

Figure 11-1 is a block diagram of the timer module.

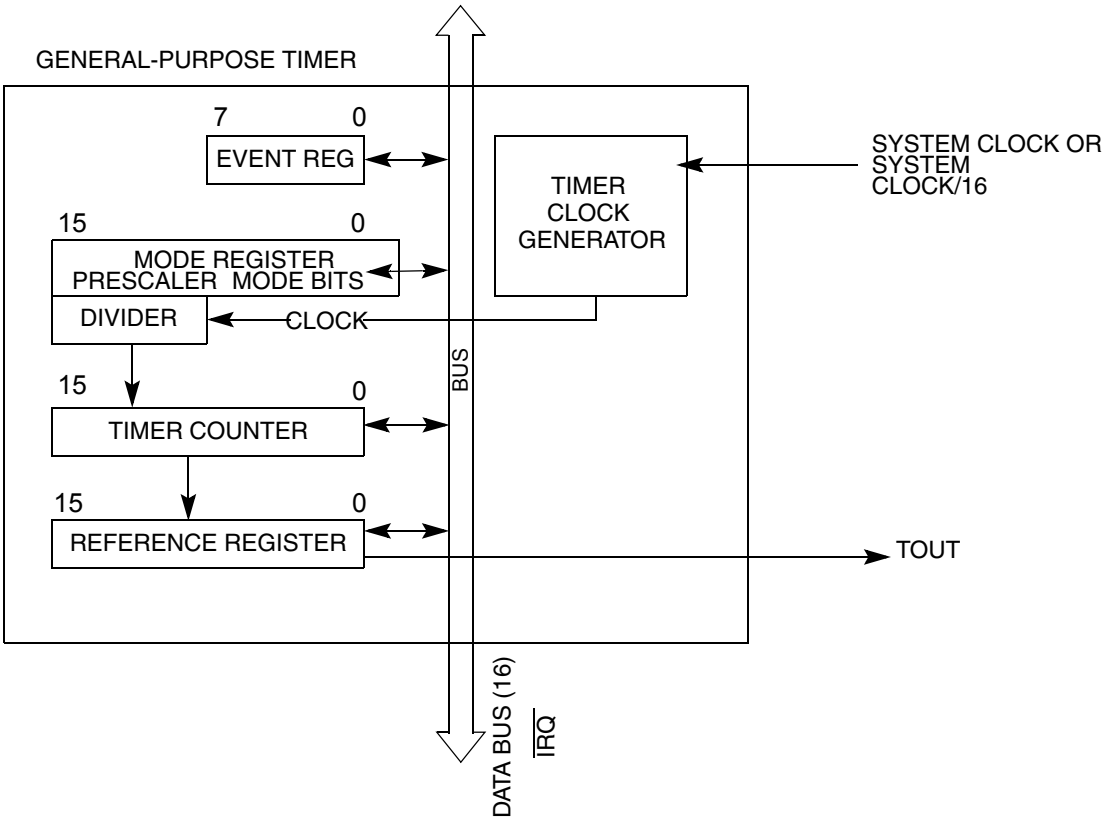


Figure 11-1. Timer Block Diagram Module Operation

11.4 Timer Signal Output

Only Timer0 has an output pin. The timer output pin is SDATAO1/TOUT0/GPO18. At reset, the function is set to SDATAO1.

11.5 Timer Operation

This section provides information to select the timer module's prescaler and to configure the timer.

11.5.1 Selecting the Prescaler

Users can select the prescalar clock from the SYSCLK (divided by 1 or by 16).
The CLK bits of the corresponding Timer Mode Register (TMR) select the clock input source. The prescaler is programmed to divide the clock input by values from 1 to 256. The prescalar output is used as an input to the 16-bit counter.

11.5.2 Configuring the Timer for Reference Compare

Users can configure the timer to count until it reaches a reference value at which time it either starts a new time count immediately or continues to run. The free run/restart (FRR) bit of the TMR selects either mode.

When the timer reaches the reference value, the REF bit in the TER register is set and issues an interrupt if the output reference interrupt (ORI) enable bit in TMR is set.

11.5.3 Configuring the Timer for Output Mode (TIMER0)

Timer0 can send an output signal on the timer output (TOUT0) pin when it reaches the reference value as selected by the output mode (OM) bit in the TMR. This signal can be an active-low pulse or a toggle of the current output under program control.

11.6 General-Purpose Timer Memory Map and Register Definitions

Users can modify the timer registers at any time. Table 11-1 shows the timer memory map.

Table 11-1. Memory Map for General-Purpose Timers

Timer 0 Address	Timer 1 Address	Timer Module Registers	
MBAR+$140	MBAR+$180	Timer Mode Register (TMRn)	
MBAR+$144	MBAR+$184	Timer Reference Register (TRRn)	
MBAR+$148	MBAR+$188	Timer Capture Register (TCRn)	
MBAR+$14C	MBAR+$18C	Timer Counter (TCNn)	
MBAR+$151	MBAR+$191	Reserved	Timer Event Register (TERn)

11.6.1 Timer Mode Registers (TMR0, TMR1)

The TMR is a 16-bit memory-mapped register. This register programs the various timer modes and is cleared by reset.

Address MBAR+$140
MBAR+$180

Access: Supervisor or User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PRESCALER VALUE (PS7–PS0)								CE1	CE0	OM	ORI	FRR	CLK1	CLK0	RESET
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-2. Timer Mode Register (TMR*n*)

Table 11-2. Timer Mode Register (TMR*n*) Field Descriptions

Bit Name	Description
15–8 PS	The Prescaler Value is programmed to divide the clock input by values from 1 to 256. The value 00000000 divides the clock by 1; the value 11111111 divides the clock by 256. Prescalar value = $[PS7 – PS0] + 1
7–6 CE	These bits have no function and should be set to 00.

Table 11-2. Timer Mode Register (TMR*n*) Field Descriptions (continued)

Bit Name	Description
5 OM	Output Mode 1 Toggle output 0 Active-low pulse for one system clock cycle (16.666 ns at 60 MHz)
4 ORI	Output Reference Interrupt Enable 1 Enable interrupt upon reaching the reference value 0 Disable interrupt for reference reached (does not affect interrupt on capture function) If ORI is set when the REF event is asserted in the Timer Event Register (TER), an immediate interrupt occurs. If ORI is cleared while an interrupt is asserted, the interrupt negates.
3 FRR	Free Run/Restart 1 Restart: Timer count is reset immediately after reaching the reference value 0 Free run: Timer count continues to increment after reaching the reference value
2–1 CLK	Input Clock Source for the Timer 11 Invalid 10 SYSCLK divided by 16 The clock source is synchronized with the timer. However, the divider is not reset to 0 when the timer is stopped, thus successive time-outs may vary slightly in length. 01 SYSCLK 00 Stops counter. After the counter is stopped, the value in the Timer Counter (TCN) register remains constant.
0 RST	The Reset Timer bit performs a software timer reset identical to that of an external reset. All timer registers take on their corresponding reset values. While this bit is zero, the other register values can still be written, if necessary. A transition of this bit from one to zero is what resets the register values. The counter/timer/prescaler is not clocked unless the timer is enabled. 1 Enable timer 0 Reset timer (software reset)

11.6.2 Timer Reference Registers (TRR0, TRR1)

The TRR is a 16-bit register that contains the reference value that is compared with its respective, free-running timer counter (TCN), as part of the output-compare function. TRR is a memory-mapped read/write register.

TRR is set at reset. The reference value is not matched until TCN equals TRR, and the prescaler indicates that the TCN should be incremented again. Thus, the reference register is matched after (TRR+1) time intervals.

Address MBAR+$144
MBAR+$184

Access: Supervisor or User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	REFERENCE COMPARE VALUE (REF15–REF0)															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 11-3. Timer Reference Register (TRRn)

11.6.3 Timer Counters (TCN0, TCN1)

TCN is a memory-mapped 16-bit up counter that users can read at any time. A read cycle to TCN yields the current timer value and does not affect the counting operation.

A write of any value to TCN causes it to reset to all zeros.

Address MBAR+$14C
MBAR+$18C

Access: Supervisor or User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	16-BIT TIMER COUNTER VALUE (COUNT15–COUNT0)															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-4. Timer Counter (TCN*n*)

11.6.4 Timer Event Registers (TER0, TER1)

The TER is an 8-bit register that reports events the timer recognizes. When the timer recognizes an event, it sets the appropriate bit in the TER, regardless of the corresponding interrupt-enable bits (ORI and CE) in the TMR.

TER appears as a memory-mapped register and can be read at any time.

Writing a one to a bit will clear it (writing a zero does not affect the bit value); more than one bit can be cleared at a time. The REF and CAP bits must be cleared before the timer will negate the IRQ to the interrupt controller. Reset clears this register.

Address MBAR+$151
MBAR+$191

Access: Supervisor or User read/write

	7	6	5	4	3	2	1	0
R							REF	CAP
W								
Reset	0	0	0	0	0	0	0	0

Figure 11-5. Timer Event Register (TER*n*)

Table 11-3. Timer Event Register (TER*n*) Field Descriptions

Field	Description
7–2	Reserved for future use. These bits are currently 0 when read.
1 REF	If a one is read from the Output Reference Event bit, the counter has reached the TRR value. The ORI bit in the TMR enables the interrupt request caused by this event. Writing a one to this bit will clear the event condition.
0 CAP	Not applicable

11.6.5 Timer Initialization Example Code

There are two timers on the MCF5251. With a 70 MHz clock, the maximum period is 3.83 seconds and a resolution of 14.3 ns. The timers can be free running or count to a value and reset. The following examples set up the timers:

Timer0 will count to $AFAF, toggle its output, and reset back to $0000. This will continue infinitely until the timer is disabled or a reset occurs. No interrupts are set. Prescale is set at 256 and the system clock is divided by 16, therefore resolution is (16 × (256))/70 MHz = 58.51us. Timeout period is (16 × 256 × 44976)/70 MHz = 2.63s ($0 - $AFAF = 44976 decimal).

NOTE

The timers were initialized in the SIM to have interrupt values. The following examples have the interrupts disabled. The initialization in the SIM configuration was for reference. The Timers CANNOT provide interrupt vectors, only autovectors.

Autovectors and ICRs have been set up as follows. The interrupt levels and priorities were chosen by random for demonstrative purposes. Users should define the interrupt level and priorities for their specific application.

11.6.5.1 Timer0 (Timer Mode Register)

Bits 15:8 sets the prescale to 256 ($FF)

Bits 7:6 set for no interrupt (“00”)

Bits 5:4 sets output mode for “toggle”. No interrupts(“10”)

Bits 3 set for “restart” (“1”)

Bits 2:1 set the clocking source to system clock/16 (“10”)

Bit 0 enables/disables the timer (“0”)

```
move.w #$FF2C,D0;Setup the Timer mode register (TMR0)
move.w D0,TMR1;                    Bit 1 is set to 0 to disable the timer
move.w #$0000,D0; writing to the timer counter with any value resets it to zero
move.w D0,TCN1;
```

11.6.5.2 Timer0 (Timer Reference Register0)

The TRR register is set to $AFAF. The timer will count up to this value (TCN = TRR), toggle the “TOUT” pin, and reset the TCN to $0000.

```
move.w #$AFAF,D0;Setup the Timer reference register (TRR0)
move.w D0,TRR1
```

Other registers used for TIMER 0

TCR0;TIMER0 Capture Register, 16-bit, R

TER0;TIMER0 Event Register, 8-bit, R/W

Chapter 12
Analog to Digital Converter (ADC)

This chapter explains the ADC operation, memory map and register descriptions, and setup recommendations of the ADC to external components.

12.1 Overview

The ADC functionality is based on the sigma-delta concept using 12-bit resolution with a measurement frequency of ADCLK / 4096.

12.1.1 Block Diagram

The ADC block diagram and external circuit example is shown in the below figure.

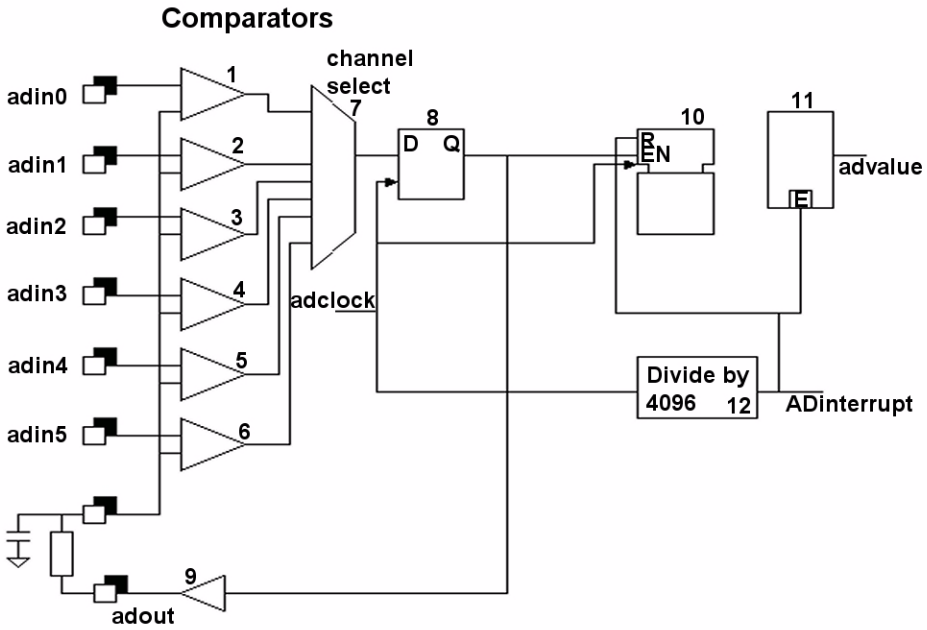


Figure 12-1. ADC Block Diagram and External Components

12.2 External Signal Description

The ADC has six muxed inputs with the following pin names.

1. ADIN0/GPI52
2. ADIN1/GPI53
3. ADIN2/GPI54

4. ADIN3/GPI55
5. ADIN4/GPI56
6. ADIN5/GPI57

The ADOUT signal on the ADOUT/SCLK4/GPIO58 pin provides the ADC comparators (ramping) reference voltage in PWM format. This output requires an external integrator circuit (resistor/capacitor) to convert the PWM source to a DC level which is then input to the ADREF pin. A circuit example is shown in Figure 12-1.

Only one ADC input can be converted at any one time. The input to be converted is selected via the source select bits of the ADconfig register. An interrupt can be provided when the ADC measurement cycle is complete.

12.3 ADC Memory Map and Register Definitions

This section discusses the two user-accessible ADC registers, ADconfig and ADvalue.

Table 12-1. ADC Memory Map

MBAR2 Offset	Register	Width	Access	Reset Value	Section/Page
0x402	ADconfig—AD configuration register	16	R/W	Undefined	12.3.1/12-2
0x406	ADvalue—AD value register	16	R	Undefined	12.3.2/12-3

12.3.1 AD Configuration Register (ADconfig)

The device will select one of the six inputs using multiplexer (**7**) as shown in Figure 12-1. ADOUT is calculated via flip-flop (8) and buffer (9). The feed-back loop (1-7-8-9) will keep the voltage on the external integrator capacitor close to ADIN0, and in this way, the voltage on ADIN0 is proportional to the duty cycle of the signal on ADOUT.

The circuit will measure the duty cycle of the ADOUT signal. Every time ADOUT is high, counter (10) will increment. Every 4096 AD_CLK clock pulses the value from counter (10) is latched into register (11), and ADInterrupt is generated. Counter (10) is also reset.

On reception of ADInterrupt, the processor will read ADvalue(12:0) from the ADvalue register. This value is in range 0-4096, and indicates duty cycle of ADOUT. See Figure 12-2 for illustration of valid bits in the ADconfig register and Table 12-2 for description of the bit fields.

Address MBAR2 + 0x402 (ADCONFIG) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R						Source Select			INTCLR	INTEN	ADOUT_DRIVE		ADCLK_SEL			
W									w1c							
Reset	–	–	–	–	–	0	0	0	–	–	–	–	–	–	–	–

Figure 12-2. AD Configuration Register (ADconfig)

Table 12-2. ADconfig Register Field Descriptions

Field	Description
15–11	Reserved, should be cleared.
10–8 Source Select	ADC source select. 000 ADIN0 001 ADIN1 010 ADIN2 011 ADIN3 100 ADIN4 101 ADIN5 **Note:** Only one channel can be measured at any one time.
7 INTCLR	ADC interrupt pending. Indicates that an interrupt is pending. Write one to clear; writing zero has no affect. 0 No ADC interrupt pending 1 ADC Interrupt pending
6 INTEN	ADC interrupt enable. 0 Interrupt disabled 1 Interrupt enabled
5–4 ADOUT_DRIVE	00 ADOUT drives +Vdd for Hi, GND for low 01 ADOUT tri-state 10 ADOUT drives HI_Z for Hi, GND for low 11 ADOUT drives +Vdd for Hi, HI_Z for low **Note:** For the circuit shown in Figure 12-1, the ADOUT_DRIVE should be set to 00. Other circuits can use settings 10 or 11.
3–0 ADCLK_SEL	ADCLK source select. Selects the clock source for the ADC as a function of BUSCLK. 0000 BUSCLK 0001 BUSCLK / 2 0010 BUSCLK / 4 0011 BUSCLK / 8 0100 BUSCLK / 16 0101 BUSCLK / 32 0110 BUSCLK / 64 0111 BUSCLK / 128 1000 BUSCLK / 256 Else Reserved

12.3.2 AD Value Register (ADvalue)

See Figure 12-2 for illustration of valid bits in the ADvalue Register and Table 12-2 for description of the bit fields.

Address MBAR2 + 0x406 (ADVALUE) Access: User read-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R				OF	ADVALUE											
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 12-3. AD Value Register (ADvalue)

Table 12-3. ADvalue Register Field Descriptions

Field	Description
15–13	Reserved, should be cleared.
12 OF	Overflow. Indicates the input voltage is out of range. The ADC block does not support full rail-to-rail conversions. 0 No overflow condition. 1 Overflow. Input signal is outside the operating voltage range of the ADC.
11–0 ADVALUE	AD measurement result.

12.4 Functional Description

Each ADC input channel has its own on-chip comparator the output of which is multiplexed with the digital section.The reason to have separate comparators for each channel allows for the inputs to be used as GPI's. In this mode the ADREF should be a fixed level (typically VDD/2) and each comparator is then being used to indicate if its input is above or below this reference (HIGH or LOW). The state of each GPI in this case is read using the GPIO_READ registers.

NOTE

It is possible to mix the use of each of these inputs between ADC and GPI function as the ADOUT/SCLK4/GPIO58 pin can be switched between providing the ramping ADOUT signal (ADC mode) to providing a fixed level (VDD/2) by switching its operation to SCLK4 mode when appropriate. SCLK4 will output a 50% duty cycle clock. Which when integrated will produce a reference voltage close to VDD/2. The output frequency of SCLK4 can be varied by programming the IIS4 Audio register. See Section 17.5, "Serial Audio Interface (I2S/EIAJ) Register Descriptions."

The ADC uses the sigma-delta modulation principle. The ADC external components required are an integrator circuit comprising of a resistor and capacitor. The desired values for this integrator network are dependent on the BUSCLK clock frequency and the associated setting of the ADconfig[ADCLK_SEL] bits, which determine the maximum ADOUT PWM frequency.

12.4.1 Recommendations to Set-up of ADC and External Components

Do not run the ADC clock any faster than 10 MHz. This results in a maximum sampling frequency of 2441 Hz (10 MHz/4096).

To calculate the external component values use the following equation:

$$RC = K \times t \qquad \textit{Eqn. 12-1}$$

where K is a constant. If K is small, the ripple on the comparator input will be quite large, and there will be some mis-measurement because the average value on both comparator pins is not equal. If K is small, the comparator will have difficulty determining if the result is negative or positive. The circuit becomes sensitive to noise. Therefore, we recommend to set K between 20 and 50.

$$t = \frac{1}{ADCCLOCK}$$ *Eqn. 12-2*

The ADCCLOCK should be such that the comparator can take the decision whether its output needs to be positive or negative in time t.

When K is high it means that RC is high relative to the clock period of the ADC and the voltage step over C per clock cycle becomes small. Therefore, the ADC becomes slow in responding to changes of the input voltage and this affects the accuracy of the measurement.

For a correct measurement, the voltage over C should be equal to the input voltage during the entire measurement cycle (1024 ADC clocks). Therefore for the first conversion or when switching channels, two consecutive measurements should be made and the first one ignored. This then allows the capacitor (C) to charge to the average value of the channel. If voltages between the channels are significantly different, the first measurement will be inaccurate because the capacitor may not have charged to the new level in time. When reading the same channel, it is not necessary to ignore every other measurement.

We therefore recommend to use R = 33kΩ, C = 10nF with ADCLK = BUSCLK/256. This should produce good results for typical system clock frequencies between 30 MHz and 70 MHz.

Chapter 13
IDE and Flash Media Interface

This chapter describes the operation of the bus interface to IDE and Flash Media, the interface setup, timing and operation are provided as well as commonly used commands.

13.1 IDE and SmartMedia Overview

The MCF5251 memory bus allows connection of an IDE hard disk drive or SmartMedia flash card with a minimum of external hardware. Figure 13-1 shows the bus set-up for the MCF5251 device. The figure illustrates an interface with both an IDE device and a SmartMedia device connected although both cannot be supported simultaneously as the IDE_DIOR and IDE_DIOW signals can only be used to interface to one or the other.

NOTE

SmartMedia refers to Flash memory cards such as Compact Flash. For other Flash Media such as Secure Digital (SD), MultiMedia Card (MMC) or Memory Stick, refer to Section 13.4, “Flash Media Interface.”

For support as a Flash Media Interface, this solution is recommended.

This IDE and Flash Media Interface is not recommended for new IDE designs. All new IDE designs should use the ATA Interface, refer to Chapter 22, “USB, ATA DMA, and Clock Integration Module”. Older hard drive IDE designs may still be supported through this legacy IDE Interface.

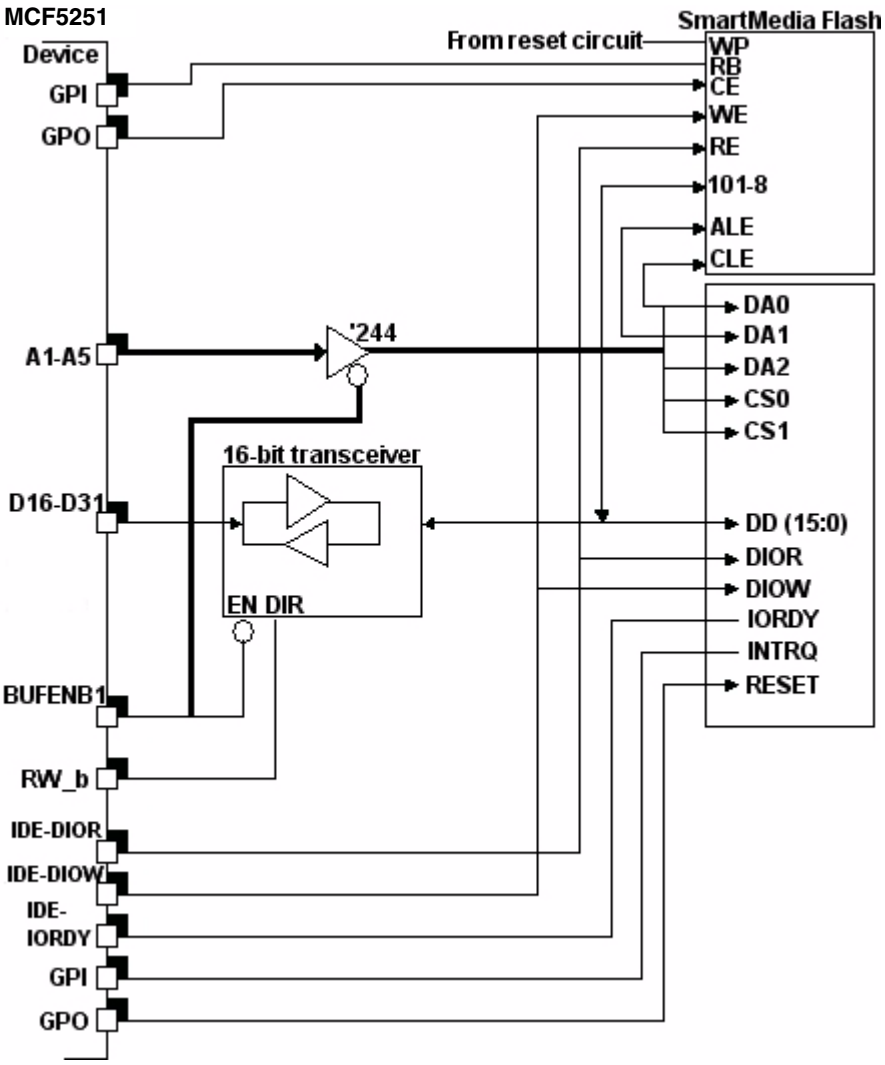


Figure 13-1. Bus Setup with IDE and SmartMedia Interface

In this example there is only one buffer between the MCF5251 memory bus and the IDE / SmartMedia interface. The SDRAM (if used) is connected directly to the memory bus along with the Flash memory (if used). The buffer therefore provides isolation (and signal buffering) between the memory bus components and the slow, high capacitance and low impedance IDE bus. Thus allowing access to the SDRAM at the highest memory bus speed (70MHz) possible. The buffer also prevents the SDRAM and Flash ROM signals from going to/from the IDE / SmartMedia interfaces.

In some systems where the Flash ROM load may be excessively high or there is the requirement for additional devices on the memory bus such as an additional SRAM or Ethernet controller. It maybe necessary to provide further isolation and buffering of the memory bus between the MCF5251 / SDRAM. There is provision for an additional buffer control signal in the system. The “first” bus buffer isolates the MCF5251 / SDRAM bus from the flash ROM and any other additional devices (SRAM, Ethernet Controller, etc.). The “second” bus buffer prevents the flash ROM signals from going to/from IDE and SmartMedia interfaces. The IDE and SmartMedia interfaces share most signals with the ColdFire address and data bus.

To support this bus set-up, a number of signals are available.

- $\overline{\text{BUFENB1}}$—active-low external buffer enable. This enable is always active when the $\overline{\text{CS0}}$/$\overline{\text{CS4}}$ pin is active, and should enable a buffer going to the external boot Flash / ROM and any additional memories or controllers.
- When $\overline{\text{CS0}}$ is being used internally as the Chip Select for the boot ROM, the $\overline{\text{CS0}}$/$\overline{\text{CS4}}$ pin operates with the $\overline{\text{CS4}}$ settings, $\overline{\text{BUFENB1}}$ settings then apply to $\overline{\text{CS4}}$.
- $\overline{\text{BUFENB2}}$—active-low external buffer enable. This enable is always inactive when the $\overline{\text{CS0}}$/$\overline{\text{CS4}}$ pin is active, and should enable a buffer for the IDE / SmartMedia Interface.
- $\overline{\text{IDE\_DIOR}}$, $\overline{\text{IDE\_DIOW}}$—active-low IDE bus read and write strobe can also be used to implement a SmartMedia interface.
- IDE_IORDY—active-high “ready” indication from IDE device to MCF5251.

NOTE

Either of the buffer enables can be programmed to be active on $\overline{\text{CS1}}$ or $\overline{\text{CS2}}$

The extra bus signals, and their configuration are detailed in the following section.

13.1.1 Buffer Enables $\overline{\text{BUFENB1}}$, $\overline{\text{BUFENB2}}$, and Associated Logic

Buffer enables $\overline{\text{BUFENB1}}$ and $\overline{\text{BUFENB2}}$ allow a seamless interface to external bus buffers. The buffers are placed on the address and the data bus.

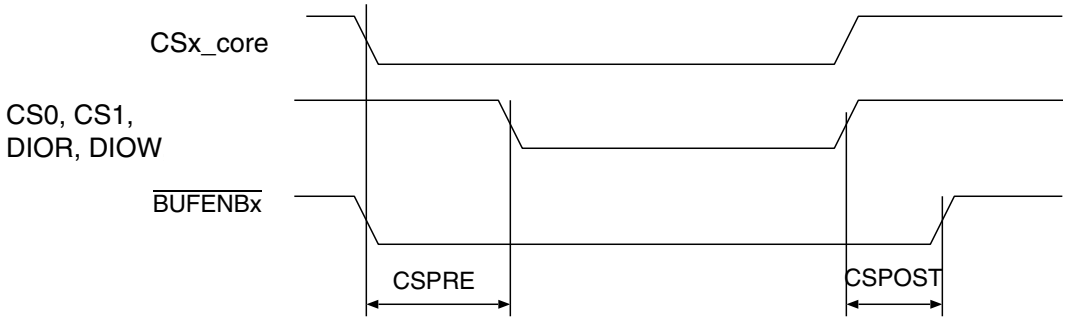


Figure 13-2. Buffer Enables ($\overline{\text{BUFENB1}}$ and $\overline{\text{BUFENB2}}$)

As shown in Figure 13-2, the buffer enables $\overline{\text{BUFENB1}}$ and $\overline{\text{BUFENB2}}$ will go active at time CSPRE before the falling edge of the Chip Select signal, and continue to be active for a time CSPOST after the rising edge of the chip select signal. The pre-drive time CSPRE is realized by delaying the falling edge of the select signal. If pre-drive time CSPRE is programmed non-zero, and internal ColdFire cycle termination is used, chip select length will be CSPRE shorter than the programmed length. Times CSPRE, CSPOST are the same for both $\overline{\text{BUFENB1}}$ AND $\overline{\text{BUFENB2}}$. Times CSPRE, CSPOST are independently programmable for every Chip Select.

Buffer enable configuration is programmable using the IDE_CONFIG1 register.

Table 13-1. IDEConfig1 Register

Address	Access	Size Bits	Name	Description
MBAR2 + 0x18c	RW	32	IDE CONFIG1	Configuration of buffer enable generation

Table 13-2. IDEConfig1 Register Field Descriptions

Field	Description	Res
2–0 CS0/CS4PRE	Pre-drive for $\overline{CS0}$/$\overline{CS4}$ 000 No predrive 0011 clock 0102 clocks 0113 clocks 1004 clocks 1015 clocks	0
4–3 CS0/CS4POST	Post-drive for $\overline{CS0}$/$\overline{CS4}$ 00 No post-drive 01 1 clock post-drive 10 2 clock post drive 11 3 clock post drive	0
7–5 cs1pre	Pre-drive for $\overline{CS0}$/$\overline{CS4}$ 000 No predrive 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks	0
9–8 cs1post	Post-drive for $\overline{CS1}$ 00 No post-drive 01 1 clock post-drive 10 2 clock post drive 11 3 clock post drive	0
12–10 cs2pre	Pre-drive for $\overline{IDE\_DIOR}$, $\overline{IDE\_DIOW}$ 000 No predrive 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks	0
14–13 cs2post	Post-drive for $\overline{CS2}$ 00 No post-drive 01 1 clock post-drive 10 2 clock post drive 11 3 clock post drive	0
16 bufen1cs1en	0 $\overline{BUFENB1}$ inactive on $\overline{CS1}$ cycles 1 $\overline{BUFENB1}$ active on $\overline{CS1}$ cycles	0
17 bufen1cs2en	0 $\overline{BUFENB1}$ inactive on $\overline{IDE\_DIOR}$, $\overline{IDE\_DIOW}$ cycles 1 $\overline{BUFENB1}$ active on $\overline{IDE\_DIOR}$, $\overline{IDE\_DIOW}$ cycles	0

Table 13-2. IDEConfig1 Register Field Descriptions (continued)

Field	Description	Res
18 bufen1cs3en	0 $\overline{\text{BUFENB1}}$ inactive on $\overline{\text{CS3}}$ cycles 1 $\overline{\text{BUFENB1}}$ active on $\overline{\text{CS3}}$ cycles	0
19 bufen2cs1en	0 $\overline{\text{BUFENB2}}$ inactive on $\overline{\text{CS1}}$ cycles 1 $\overline{\text{BUFENB2}}$ active on $\overline{\text{CS1}}$ cycles	0
20 bufen2cs2en	0 $\overline{\text{BUFENB2}}$ inactive on $\overline{\text{IDE\_DIOR}}$, $\overline{\text{IDE\_DIOW}}$ cycles 1 $\overline{\text{BUFENB2}}$ active on $\overline{\text{IDE\_DIOR}}$, $\overline{\text{IDE\_DIOW}}$ cycles	0
21 bufen2cs3en	0 $\overline{\text{BUFENB2}}$ inactive on $\overline{\text{CS3}}$ cycles 1 $\overline{\text{BUFENB2}}$ active on $\overline{\text{CS3}}$ cycles	0
24–22 cs3pre	Pre-drive for $\overline{\text{CS3}}$ 000 no predrive 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks	0
26–25 cs3post	Post-drive for $\overline{\text{CS3}}$ 00 no post-drive 01 1 clock post-drive 10 2 clock post drive 11 3 clock post drive	0
27 DIOR on write	0 $\overline{\text{IDE\_DIOR}}$ not active during write cycles 1 $\overline{\text{IDE\_DIOR}}$ active during write cycles	0
28 N/A	N/A	0

13.1.2 Generation of $\overline{\text{IDE\_DIOR}}$ and $\overline{\text{IDE\_DIOW}}$

$\overline{\text{IDE\_DIOR}}$ and $\overline{\text{IDE\_DIOW}}$ are created by gating $\overline{\text{CS2}}$ with $\overline{\text{RW}}$.

$\overline{\text{IDE\_DIOR}}$ is programmable to go active on write cycles. It therefore can be used as an extra Chip Select ($\overline{\text{CS2}}$), if required.

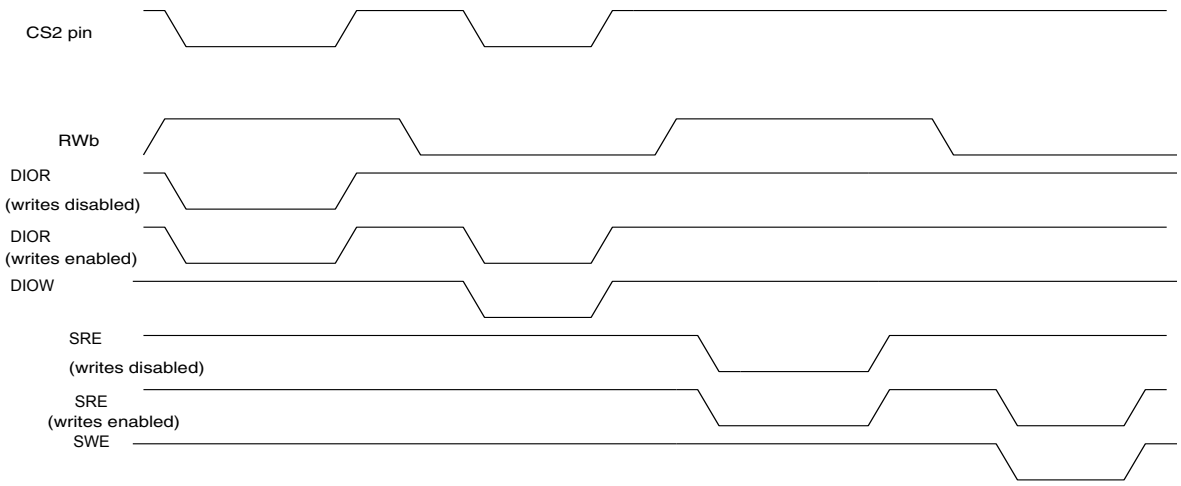


Figure 13-3. IDE_DIOR Timing Diagram

13.1.3 Cycle Termination on $\overline{CS2}$ ($\overline{IDE\_DIOR}$, $\overline{IDE\_DIOW}$)

Dedicated logic has been added to the MCF5251 to allow IDE compliant cycles on the bus. The logic can generate the transfer acknowledge ($\overline{TA}$) signal for $\overline{CS2}$ access. The manner in which the $\overline{TA}$ signal is generated is programmable using the IDE config 2 register, and is compatible with IDE/SmartMedia requirements.

Table 13-3. IDEConfig2 Register

Address	Access	Size Bits	Name	Description
MBAR2 + 0x190	RW	32	IDE config2	Configuration of $\overline{TA}$ generation on $\overline{CS2}$

Table 13-4. IDEConfig2 Register Field Description

Field	Description	RES
19 IORDY ENABLE 2	1 Allow IORDY to delay $\overline{TA}$ generation for $\overline{CS2}$ 0 Do not look at IORDY for $\overline{CS2}$ $\overline{TA}$ generation	0
18 TA ENABLE 2	1 Generate $\overline{TA}$ for $\overline{CS2}$ accesses 0 Do not generate $\overline{TA}$ for $\overline{CS2}$	0
17 IORDY ENABLE 3	Reserved set to 0	0
16 TA ENABLE 3	Reserved set to 0	0
15–8 WAITCOUNT2	CS2 delay count. Controls $\overline{TA}$ timing for read cycles	0
7–0 WAITCOUNT3	CS2 delay count. Controls $\overline{TA}$ timing for write cycles	0

The timing diagram for a non-IORDY controlled IDE/SmartMedia $\overline{TA}$ generation is shown in Figure 13-4

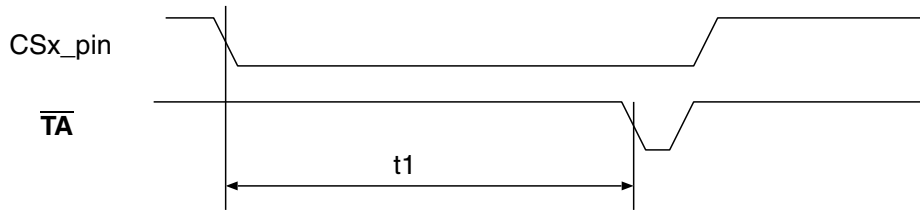


Figure 13-4. Non-IORDY Controlled IDE/SmartMedia TA Timing

The system also supports dynamic lengthening of CS2 (IDE_DIOR, IDE_DIOW) cycles using the IDE_IORDY signal. The timing diagram is shown in Figure 13-5.

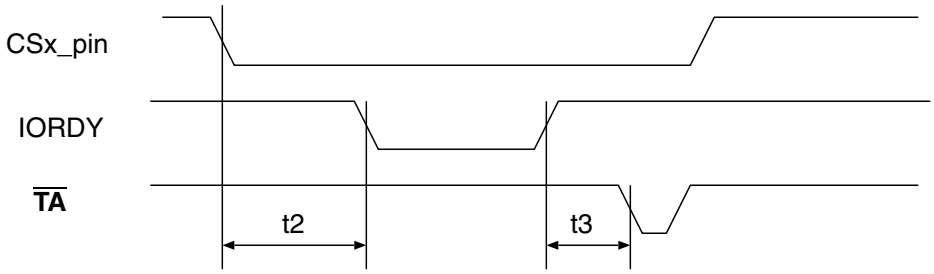


Figure 13-5. CS2 (IDE_DIOR, IDE_DIOW)

Table 13-5. IDE_DIOR, IDE_DIOW, and IDE_IORDY Timing Parameters

Timing Parameter	Description	Min	Typ	Max
t1	IDE_DIOR, IDE_DIOW low to TA	–	Read (waitCount2 + 2.5)T Write (waitCount3 + 2.5)T	–
t2[1]	IDE_DIOR, IDE_DIOW low to IDE_IORDY low	0 0	–	Read (waitCount2 + 1.5)T Write (waitCount3 + 1.5)T
t3	IDE_IORDY high to TA	2T	–	3T

[1] t2 is relevant for IDE_IORDY controlled cycles only.

13.2 SmartMedia Interface Setup

The SmartMedia block must be connected to the bus as follows:

- RE input connect to MCF5251 IDE_DIOR output
- WE input connect to MCF5251 IDE_DIOW output
- D0–7 connect to MCF5251 data bus wires 31–24
- CE connect to always low
- ALE connect to general purpose output
- CLE connect to general purpose output
- R/B connect to general purpose input

NOTE

A SmartMedia interface and an IDE interface cannot be implemented simultaneously in the same hardware application as they both share the same read and write strobe signals on the MCF5251.

To set up the SmartMedia interface perform the following tasks.

1. Program the three Chip Select registers inside the chip select modules (CSAR2, CSMR2, CSCR2) as follows:
   - CSCR2 bit fields must be programmed as follows:
     - AA—0 ($\overline{TA}$ signal generated by IDEconfig2 register logic)
     - WS[3:0]—not relevant
     - PS[1:0]—01 (8 bit port size)
     - BSTR, BSTW—00 (no burst read/write cycles)
2. Program the IDE config1 register. Only fields CS2PRE, CS2POST, BUFEN1CS2EN, BUFEN2CS2EN. The values required for the buffer enable signals BUFEN1CS2EN and BUFEN2CS2EN depend on the hardware configuration. If two buffers are used in cascade, both bits must be 1. Fields CS2PRE and CS2POST are relevant and are detailed later in this section.
3. Program the IDE config2 register as follows:
   - $\overline{TA}$ enable 2 = '1.'
   - IDE_IORDY enable 2 = '0.'
   - WAITCOUNT2 is required and is explained later in this section.

13.2.1 SmartMedia Timing

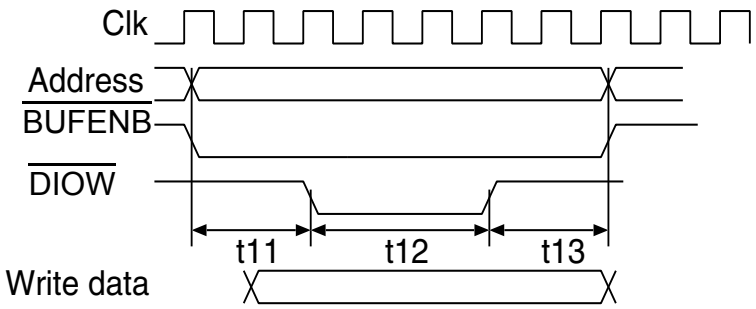


Figure 13-6. SmartMedia Timing

Table 13-6. SmartMedia Timing Values

SmartMedia Timing Symbol	Typical Value nS	Controlled by Setting	Equation (Approximately)	Comment
tCLS, tCLH, tALS, tALH	20, 40	–	CS2PRE > t1 - tbuf	Realized in software because CLE and ALE are driven by gpio.

Table 13-6. SmartMedia Timing Values (continued)

SmartMedia Timing Symbol	Typical Value nS	Controlled by Setting	Equation (Approximately)	Comment
tREA	45	WAITCOUNT2	(waitCount2 + 3.5)T > tREA	–
tDH	20	CS2POST	CS2POST > tDH	To meet this timing, typical value for cs2post is 20 ns

Under typical circumstances, CS2PRE = 0 clocks, waitCount2 = 1 or 2.

NOTE

If CS2POST is set to 2, every write cycle is lengthened with 1 clock. If CS2POST is set to 3, every write cycle is lengthened with 2 clocks.

13.3 Setting Up The IDE Interface

To set up the IDE interface, complete the following tasks.

NOTE

A SmartMedia interface and an IDE interface cannot be implemented simultaneously in the same hardware application as they both share the same read and write strobe signals on the MCF5251.

1. Program the Chip Select 2 registers inside the chip select modules. (CSAR2, CSMR2, CSCR2).
   — CSAR2, CSMR2 must be programmed to see the IDE interface in the correct part of the ColdFire address map.
   — CSCR2 bit fields must be programmed as follows:

- AA—0 ($\overline{TA}$ signal generated by IDECONFIG2 register logic)
- WS[3:0]—not relevant
- PS[1:0]—10 (16 bit port size)
- BSTR, BSTW—00 (no burst read/write cycles)

2. Program the IDE config1 register. Fields CS2PRE, CS2POST, BUFEN1CS2EN, BUFEN2CS2EN, and DIOR active during write are relevant. The values required for the buffer enable signals BUFEN1CS2EN and BUFEN2CS2EN depend on the hardware configuration. If two buffers are used in cascade, both bits must be 1. Fields CS2PRE and CS2POST are relevant and are explained later in this section.
3. Program IDECONFIG2 register. Program this register as follows:
   — $\overline{TA}$ enable 2 = ‘1’.
   — IDE_IORDY enable 2 = ‘1’ if IDE_IORDY is connected from the IDE drive to the MCF5251 chip.
   — IDE_IORDY enable 2 = ‘0’ if IDE_IORDY wait handshake is not used.
   — WAITCOUNT2 is required and is explained later in this section.

13.3.1 IDE Timing Diagram

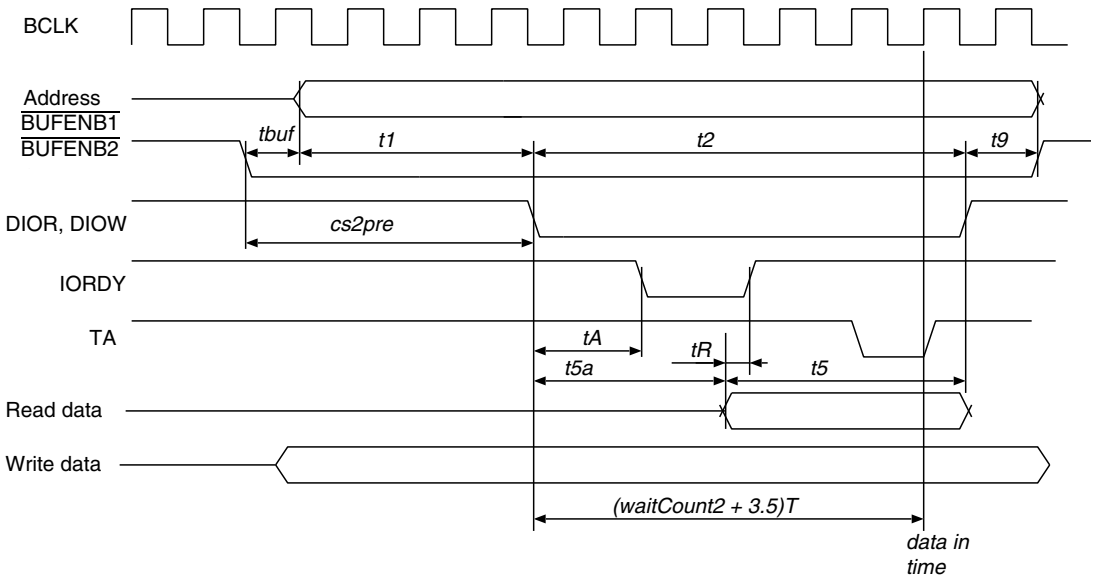


Figure 13-7. IDE Timing

Table 13-7. IDE Timing Values

ATA Timing Symbol	ATA4 Value	Controlled by Setting	Equation (Approximately)	Comment
t1	25	CS2PRE	CS2PRE > t1 - tbuf	tbuf is external buffer enable time. cs2pre must be set high enough to provide sufficient address-to-$\overline{\text{DIOR}}$, $\overline{\text{DIOW}}$ setup time. Typical cs2pre values will range from 3 to 5 SCLK clocks.
t2	70	WAITCOUNT2	(WAITCOUNT2+ 4)T > t2	t2 is the $\overline{\text{DIOR}}$, $\overline{\text{DIOW}}$ low period. To meet 70 nS t2 period, waitCount2 must be set to 3.
t5a	50	WAITCOUNT2	(WAITCOUNT2 + 3.5)T > t5a + tio + tbuf	tio = Input/output delay of device. Typ. 10 nS tbuf = External buffer delay. Typ. 15 nS To meet this timing, waitCount2 must be set to 4-5
tA	35	WAITCOUNT2	(WAITCOUNT2 + 1.5)T > tA + tio	To meet this timing, waitCount2 must be set 3-4.
tR	0	–	3T > tbuf + tdel - tR	tdel = time difference between path from IORDY and from read data Read data in device must be valid 3 clocks after IORDY going high.
t9	10	CS2POST	CS2POST > t9	To meet this timing, typical value for cs2post is 10 nS.

NOTE

If CS2POST is set to 2, every write cycle is lengthened with 1 clock. If CS2POST is set to 3, every write cycle is lengthened with 2 clocks. This marginally reduces throughput.

NOTE

A 3-clock cycle hold time to any MCF5251 external access has been added. As a result, hold time address to TA and write data to TA is not an issue.

13.4 Flash Media Interface

The MCF5251 is capable of interfacing with Sony Memory Stick and Multi-Media Card (MMC) / Secure Digital (SD) flash cards. The interface can handle one of them at any given time, but not both at the same time.

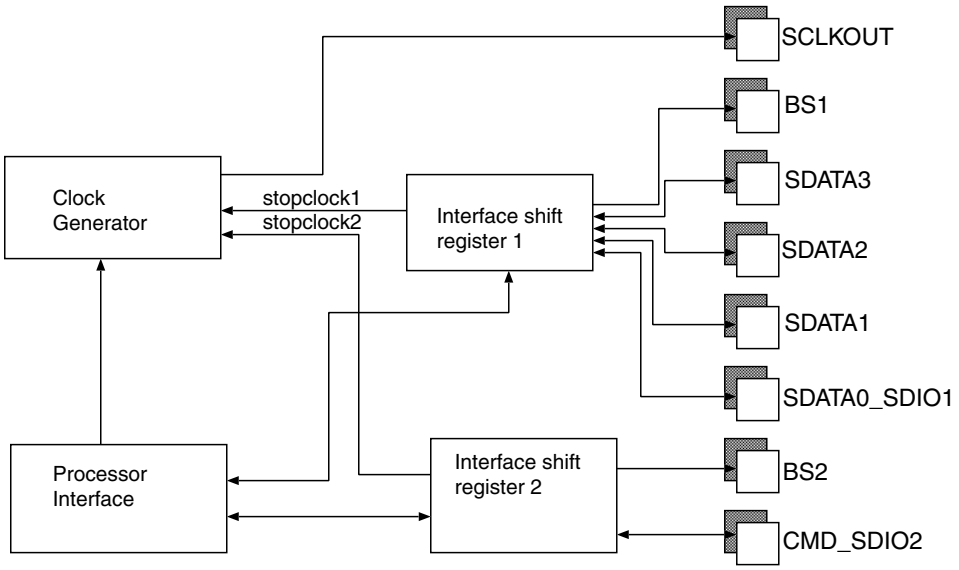


Figure 13-8. Flash Media Block Diagram

In the Flash Media interface there are four blocks:

1. The *clock generator* generates the clock to the flash device
2. The *Processor interface* handles interrupts and processor I/O
3. Interface shift register 1
4. Interface shift register 2

Each interface shift register is a serial interface to the Flash Media device. The two interfaces share the clock generating circuitry.

The flash media interface can operate in two modes.

1. *MemoryStick* mode. In this mode it is possible to connect two Sony Memory Stick cards. Each interface can handle one Memory Stick card. The two interfaces share only the clock generating logic, all other logic is fully independent.

2. *SecureDigital* mode. In this mode it is possible to connect one SD card. The SD card has a *command* line and 1 or 4 *serial data* lines. The interface shift register 1 will handle communication on the *serial data* lines, the interface shift register 2 will handle communication on the *command* line. From a software point of view, the two interfaces operate independently.

13.5 Flash Media Interface Memory Map and Register Definitions

The Flash Media interface contains eight 32-bit registers.

Table 13-8. Flash Media Registers

Address	Access	Size Bits	Name	Description
MBAR2+ 0x460	RW	32	FLASHMEDIACONFIG	Clock and general configuration
MBAR2+ 0x464	RW	32	FLASHMEDIACMD1	Command register for interface 1
MBAR2+ 0x468	RW	32	FLASHMEDIACMD2	Command register for interface 2
MBAR2+ 0x46C	RW	32	FLASHMEDIADATA1	Data register for interface 1
MBAR2+ 0x470	RW	32	FLASHMEDIADATA2	Data register for interface 2
MBAR2+ 0x474	RW	32	FLASHMEDIASTATUS	Status register
MBAR2+ 0x478	RW	32	FLASHMEDIAINTEN	Interrupt enable register
MBAR2+ 0x47C	R	32	FLASHMEDIAINTSTAT	Interrupt status register
MBAR2+ 0x47C	W	32	FLASHMEDIAINTCLEAR	Interrupt clear register

13.5.1 Flash Media Clock Generation and Configuration

Clock generation and selection of the card type is accomplished by programming the FLASHMEDIACONFIG register as shown in Figure 13-9.

Address MBAR2 + 0x460 Access: User read/write

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CLOCKCOUNT1								CLOCKCOUNT0							
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 13-9. Flash Media Configuration Register (FLASHMEDIACONFIG)

Table 13-9. Flash Media Configuration Register Field Descriptions

Field	Description	Res
31–22	Reserved	–
21–20 CARDTYPE	Card Type 00 Sony Memory Stick 01 SecureDigital, 1-bit serial data 11 SecureDigital, 4-bit serial data	0
19 RECEIVEEDGE	Receive Edge[1] 1 Receive data on falling edge of SCLKOUT pin 0 Receive data on rising edge of SCLKOUT pin	0
18	Reserved	0
17–16 STOPCLOCK	Stop Clock[3] 00 Normal operation 01 Freeze clock low 10 Freeze clock high	01
15–8 CLOCKCOUNT1	CLOCKCOUNT1+1[2, 3] is the sclk_out_pin high period in number of bus clocks	15
7–0 CLOCKCOUNT0	CLOCKCOUNT0+1[2, 3] is the sclk_out_pin low period in number of bus clocks	15

1 In SD mode, this bit should be programmed 1. In MemoryStick mode, programming 1 gives more relaxed timing, however Memory Stick specs stipulate it should be 0.

2 The clock generator will increase the length of some SCLKOUT clock cycles to avoid bus contention when the SDIO pin switches from input to output, or from output to input mode. The clock generator will stop the SCLKOUT clock if this is necessary to avoid buffer overrun or buffer underrun.

3 It is acceptable to reprogram these bits while the interface is running. No glitch will occur on sclk_out.

13.5.2 Flash Media Interface Operation

The Flash Media interface is built around two *Interface Shift Registers*, each of which work independently. Figure 13-10 shows a block diagram of one interface shift register.

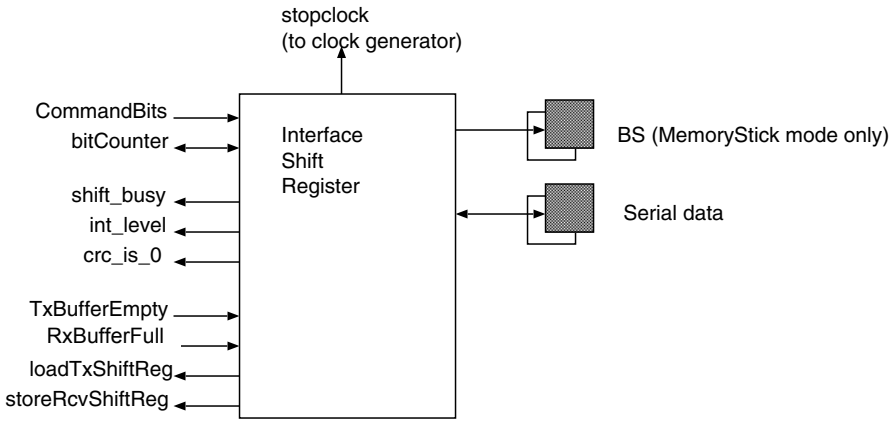


Figure 13-10. Shift Register

The processor interface sends *commands* to the interface shift register. One command instructs the interface shift register to do one of the following:

- Transmit a packet of N bits to the Flash Media device. The number of bits N is programmable. It is also programmable if bits 15:0, or bits 47:0 in SD wide bus mode, need to be replaced with a valid CRC or not. CRC insertion is possible for Memory Stick data packet and SecureDigital data packets. CRC insertion is not possible for SecureDigital command packets.
- Receive a packet of N bits from the Flash Media device. The number of bits N is programmable. After reception of all bits, the interface shift register will display on status line CRC_IS_0 if CRC check was successful or not. CRC check is done for Memory Stick data packets and for SecureDigital data packets. No CRC check is available for SD command packets.
- Wait for an interrupt event from the Flash Media device.

After writing a command to the interface shift register, the processor needs to monitor TxBUFFEREMPTY or RxBUFFERFULL, and read or write data to the interface as required.

- When the transmit shift register is empty, new data is loaded from the TxBUFFERREG. If the transmit buffer register is empty, the interface shift register will stop the SCLKOUT clock, and wait for new data to be written in the TxBUFFERREG.
- When the receive shift register is full, data is transferred to the RxBUFFERREG. If the receive buffer register is full, the interface shift register will stop the SCLKOUT clock, and wait until the RxBUFFERREG is read.
- If the number of bits in the packet to send/receive from the Flash Media is greater than 32, multiple longword transfers to the buffer register are needed. All of these, except the first, contain 32 packet bits. The last data word for the transfer always contains packet bits 31-0, even if CRC transmit or check is on.
  If e.g. a 48-bit transfer is requested to the Flash Media, the first data word will contain 16 bits, the second one will contain 32 bits. The first word is LSB aligned for receive data, MSB aligned for transmit data.
  This is also true if CRC insertion is involved. If a 4096 bit packet + 16 bit CRC need to be transmitted to the Flash Media, 129 long-word transfers are needed. The first long-word will contain packet bits 4095:4080, MSB aligned. The last longword will contain packet bits 15:0 padded with 16 zeros or ones. The padded value will be replaced with the CRC by the transmit interface (if the interface is programmed to do so).

During and after transmission of a command, the processor can monitor the Interface Shift Register status by looking at some status signals.

- SHIFT_BUSY This signal is high while the data transmission is in progress.
- INT_LEVEL During interrupt commands, a high on this signal indicates an interrupt event coming from the Flash Media.
- CRC_IS_0 After a read transmission is completed, this signal indicates if the packet CRC was 0 or not.
- BITCOUNTER. This counter indicates the number of bits still to be exchange with the Flash Media card.

13.5.2.1 Flash Media Command Registers in Memory Stick Mode

Address MBAR2 + 0x464
MBAR2 + 0x468

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R											SEND CRC	NEXT BS	CMDCODE			
W																
Reset	–	–	–	–	–	–	–	–	–	–	0	0	0	0	0	0

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

Figure 13-11. Flash Media Command Registers 1 & 2 (FLASHMEDIACMD) (MemoryStick Mode)

Table 13-10. Flash Media Command Registers 1 & 2 Field Description (MemoryStick Mode)

Field	Description
31–22	Reserved
21 SENDCRC	0 No CRC inserted 1 Packet bits 0-15 will be replaced with CRC
20 NEXT BS	Next value to output on BS pin (Memory Stick)
19–16 CMDCODE	0001 Read data (Memory Stick) 0010 Write data (Memory Stick) 1000 Wait for INT (Memory Stick)
15–0 BITCOUNTER	Write to this field the number of bits to be exchanged with the flash card. Read value is the number of bits remaining.

13.5.2.2 Flash Media Command Register 1 in Secure Digital Mode

Address MBAR2 + 0x464

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R										WIDE SHIFT	SEND CRC	NEXT BS	CMDCODE			
W																
Reset	–	–	–	–	–	–	–	–	–	0	0	0	0	0	0	0

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

Figure 13-12. Flash Media Command Register 1 (FLASHMEDIACMD) (Secure Digital Mode)

Table 13-11. Flash Media Command Register 1 Field Descriptions[1] (Secure Digital Mode)

Field	Description
31–23	Reserved
22 WIDESHIFT	0 1-bit data bus 1 4-bit data bus
21 SENDCRC	0 No CRC inserted 1 packet bits 0-15 will be replaced with CRC
20 NEXT BS	Next value to output on BS pin (Memory Stick)
19–16 CMDCODE	–
15–0 BITCOUNTER	Write to this field the number of bits to be exchanged with the flash card. Read value is the number of bits remaining.

1 The following codes are relevant for FlashMedia command register 1:
FLASHMEDIACMD1[22:16] = 0x44: wait for read, 4-bit wide
FLASHMEDIACMD1[22:16] = 0x04: wait for read, 1-bit wide
FLASHMEDIACMD1[22:16] = 0x66: write data, 4-bit wide
FLASHMEDIACMD1[22:16] = 0x26: write data, 1-bit wide
FLASHMEDIACMD1[22:16] = 0x00: receive handshake
FLASHMEDIACMD1[22:16] = 0x08: wait for busy signalling

13.5.2.3 Flash Media Command Register 2 in Secure Digital Mode

Address MBAR2 + 0x468 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R									DRIVE DATA	DRIVE CMD	SEND CRC	NEXT BS	CMDCODE			
W																
Reset	–	–	–	–	–	–	–	–	–	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	BITCOUNTER															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-13. Flash Media Command Register 2 (FLASHMEDIACMD) (Secure Digital Mode)

Table 13-12. Flash Media Command Register 2 Field Descriptions[1] (Secure Digital Mode)

Field	Description
31–22	Reserved
23 DRIVEDATA	0 Do not drive data line 1 Start driving data line after command transmission end
22 DRIVECMD	0 Do not drive command line 1 Start driving command line after receiving card status response

Table 13-12. Flash Media Command Register 2 Field Descriptions[1] (Secure Digital Mode) (continued)

Field	Description
21 SENDCRC	Reserved, should be 0
20 NEXT BS	Next value to output on BS pin (Memory Stick)
19–16 CMDCODE	–
15–0 BITCOUNTER	Write to this field the number of bits to be exchanged with the flash card. Read value is the number of bits remaining.

1 The following codes are relevant for FlashMedia command register 2:
FLASHMEDIACMD2[23:16] = 0x46: Send read data command to SD.
(drive cmd line after receiving flash status, do not drive data lines)
FLASHMEDIACMD2[23:16] = 0x40: Receive status for read data command from SD
FLASHMEDIACMD2[23:16] = 0xC6: Send write data command to SD.
(Drive cmd line after receiving flash status. Drive data line after sending command.)
FLASHMEDIACMD2[23:16] = 0xC0: Receive status for write data command from SD
FLASHMEDIACMD2[23:16] = 0x06: Send non-data command to SD
FLASHMEDIACMD2[23:16] = 0x00: Receive status for non-data command, stop driving cmd and data lines.

The commands and their meanings are described in detail later in this section.

13.5.3 Flash Media Data Registers

Address MBAR2 + 0x46C
MBAR2 + 0x470

Access: User read/write

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

Figure 13-14. Flash Media Data Registers 1 & 2 (FLASHMEDIACMD) (Secure Digital Mode)

Table 13-13. Flash Media Data Registers 1 & 2 Field Descriptions

Field	Description
31–0 RCVBUFFERREG	Read receive data from this register.
31–0 TXBUFFERREG	Data written to this register will be transmitted.

13.5.3.1 Flash Media Status Register

Address MBAR2 + 0x474 Access: User read/write

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R											INT_LEVEL2	SHIFT_BUSY2	CRC_IS_0_2	INT_LEVEL1	SHIFT_BUSY1	CRC_IS_0_1
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	0	0	0	0

Figure 13-15. Flash Media Status Register (FLASHMEDIASTAT)

Table 13-14. Flash Media Status Register Field Descriptions

Field	Description
31–6	Reserved
5 INT_LEVEL2	Interface 2 interrupt indicator. ‘1’ indicates interrupt condition, requiring attention, ‘0’ indicates no interrupt
4 SHIFT_BUSY2	Interface 2 shift status. ‘1’ indicates interface busy shifting data, ‘0’ indicates interface idle
3 CRC_IS_0_2	Interface 2 CRC status. Valid after read phase end. ‘1’ indicates CRC OK, ‘0’ indicates CRC fail
2 INT_LEVEL1	Interface 1 interrupt indicator. ‘1’ indicates interrupt condition, requiring attention, ‘0’ indicates no interrupt
1 SHIFT_BUSY1	Interface 1 shift status. ‘1’ indicates interface busy shifting data, ‘0’ indicates interface idle
0 CRC_IS_0_1	Interface 1 CRC status. Valid after read phase end. ‘1’ indicates CRC OK, ‘0’ indicates CRC fail

13.5.4 Flash Media Interrupt Register

There are 12 interrupt sources associated with the Flash Media interface. REGISTER FLASHMEDIAINTSTAT allows the viewing of pending interrupts. Register FLASHMEDIAINTEN allows the enabling of interrupts (‘1’ = enabled, ‘0’ = disabled). Some interrupts can be cleared by writing a ‘1’ to the corresponding bit of the FLASHMEDIAINTCLEAR register.

Address MBAR2 + 0x478 Access: User read/write

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R					TX2EMPTY	RCV2FULL	TX1EMPTY	RCV1FULL	INTLEVEL2RISE	INTLEVEL2FALL	SHIFTBUSY2RISE	SHIFTBUSY2FALL	INTLEVEL1RISE	INTLEVEL1FALL	SHIFTBUSY1RISE	SHIFTBUSY1FALL
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 13-16. Flash Media Interrupt Register

Table 13-15. Flash Media Interrupt Register Field Descriptions

Field	Description	Reset Interrupt	Associated Interrupt
31–12	Reserved	–	–
11 TX2EMPTY	Interrupt set if transmit buffer reg 2 empty	Write data	57
10 RCV2FULL	Interrupt set if receive buffer reg 2 full	Read data	57
9 TX1EMPTY	Interrupt set if transmit buffer reg 1 empty	Write data	58
8 RCV1FULL	Interrupt set if receive buffer reg 1 full	Read data	58
7 INTLEVEL2RISE	Interrupt set on rising edge of int_level_2	IntClear	59
6 INTLEVEL2FALL	Interrupt set on falling edge of int_level_2	IntClear	59
5 SHIFTBUSY2RISE	Interrupt set on rising edge of shift_busy_2	IntClear	59
4 SHIFTBUSY2FALL	Interrupt set on falling edge of shift_busy_2	IntClear	59
3 INTLEVEL1RISE	Interrupt set on rising edge of int_level_1	IntClear	60
2 INTLEVEL1FALL	Interrupt set on falling edge of int_level_1	IntClear	60

Table 13-15. Flash Media Interrupt Register Field Descriptions (continued)

Field	Description	Reset Interrupt	Associated Interrupt
1 SHIFTBUSY1RISE	Interrupt set on rising edge of shift_busy_1	IntClear	60
0 SHIFTBUSY1FALL	Interrupt set on falling edge of shift_busy_1	IntClear	60

13.5.5 Flash Media Interface Operation in Memory Stick Mode

Before any data exchange is possible with the Memory Stick, the FLASHMEDIACONFIG register must be written to set up the clock and the card type. After this, the card is accessed by issuing one of three possible command sequences. Each new command sent to the card, must toggle the BS line going out. The “handshake” phase of the Memory Stick can be implemented as a 16-bit read. There is no specific handshake command.

NOTE

The Flash Media interface can handle two Memory Stick cards. One is attached to the primary interface, the other to the secondary interface. There is one potential issue. If there is a buffer full or a buffer empty on one interface, the system will freeze the outgoing SCLK signal, which causes the second interface to go into a wait-state as well.

13.5.5.1 Reading Data from the Memory Stick

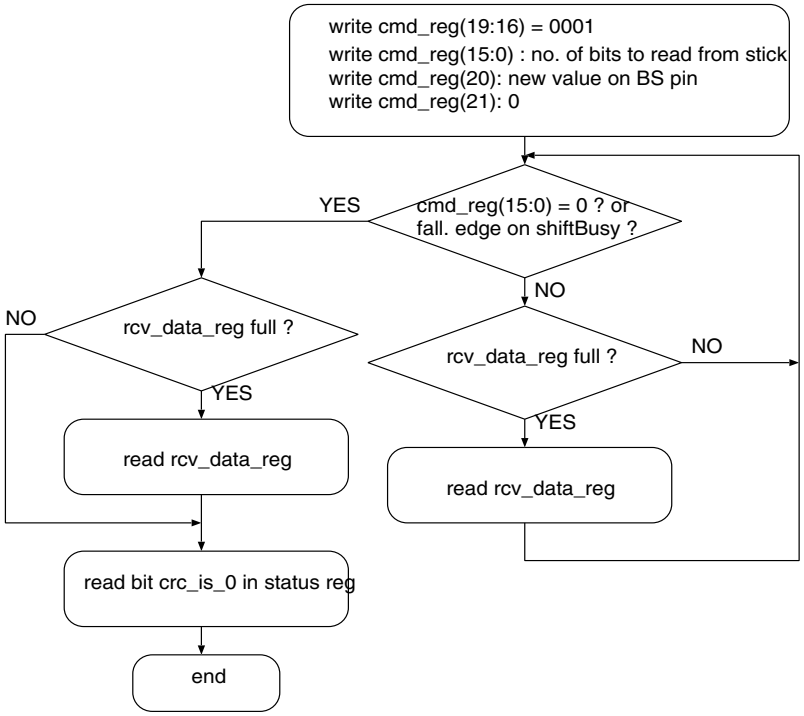
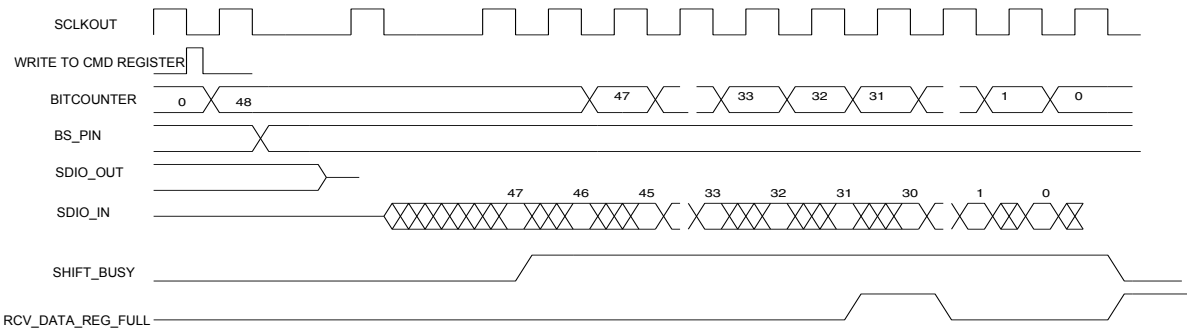


Figure 13-17. Reading Data from Memory Stick



Memory Stick interface timing diagram for cmd_reg(19:16) = 0001
(Read data from stick)

Figure 13-18. Reading Data from Memory Stick Timing

In the timing diagram, the assumption is made that the processor reads the full receive buffer register before the next 32 bits are received. If this is not the case, the Flash Media interface will stop the outgoing sclk clock, which prevents data overrun.

13.5.5.2 Writing Data to the Memory Stick

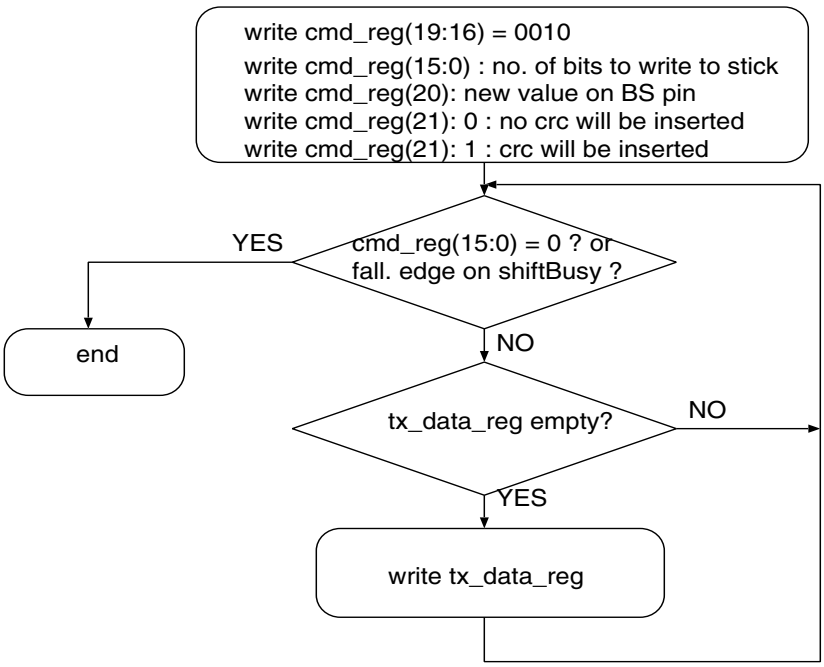
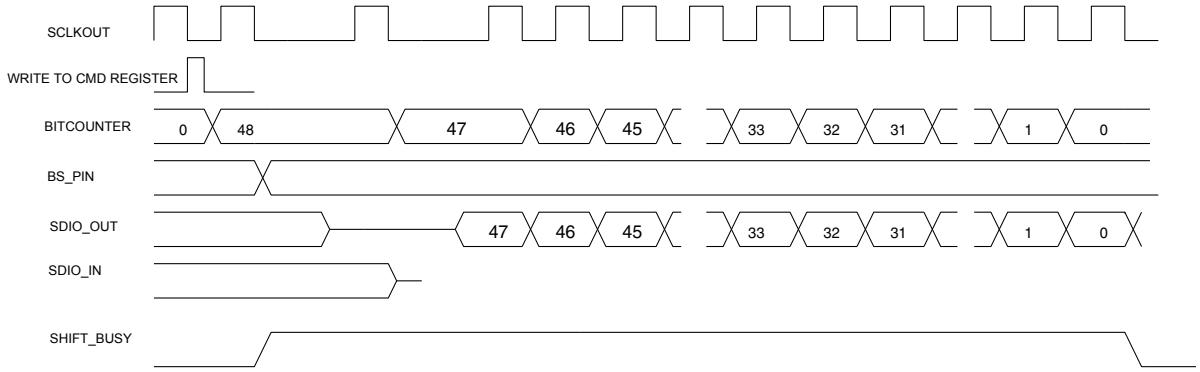


Figure 13-19. Writing Data to Memory Stick

In Figure 13-20 the assumption is made the processor writes the empty transmit buffer register before the next 32 bits are transmitted. If this is not the case, the Flash Media interface will stop the outgoing sclk clock, and in this way prevent data underrun.



Memory Stick interface timing diagram for cmd_reg(19:16) = 0010
(Write data to stick)

Figure 13-20. Writing Data to Memory Stick Timing

13.5.5.3 Interrupt from Memory Stick

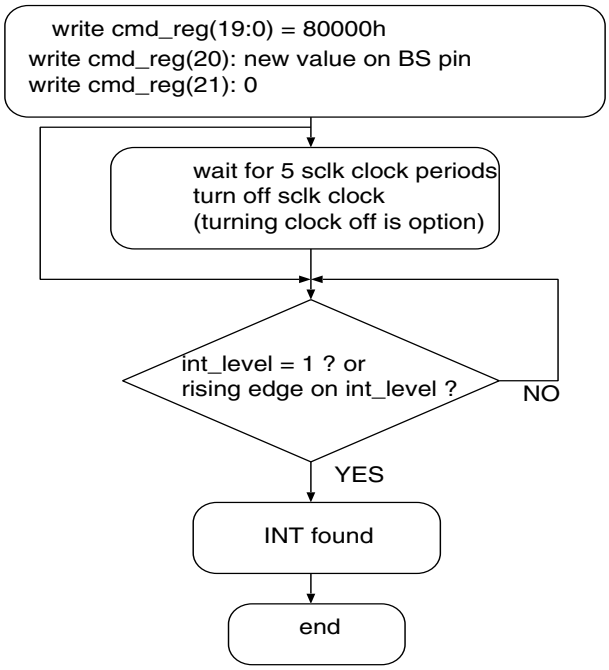
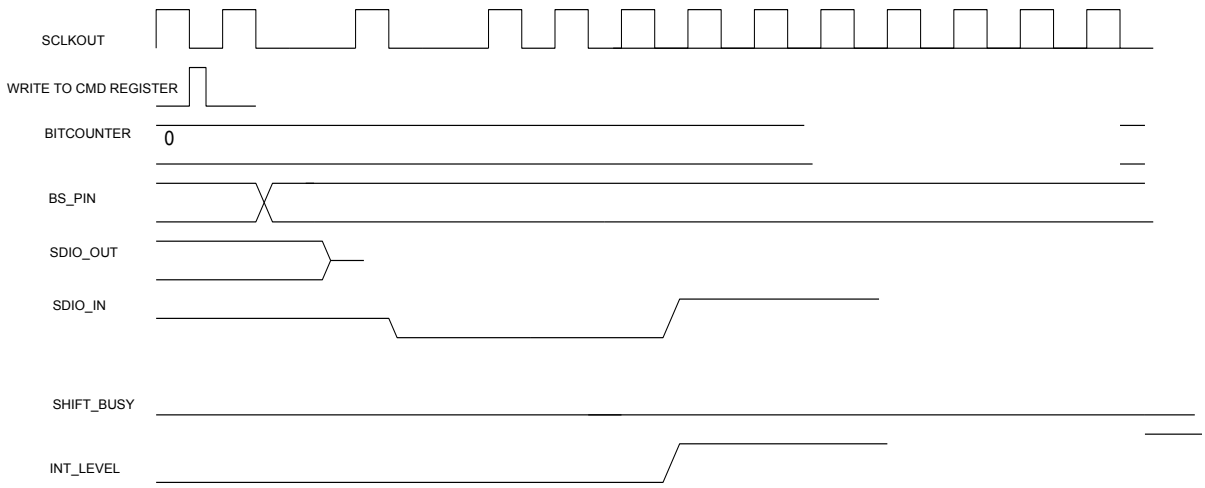


Figure 13-21. Interrupt from Memory Stick



Memory Stick interface timing diagram for cmd_reg(19:16) = 1000
(Wait for INT from stick)

Figure 13-22. Interrupt from Memory Stick

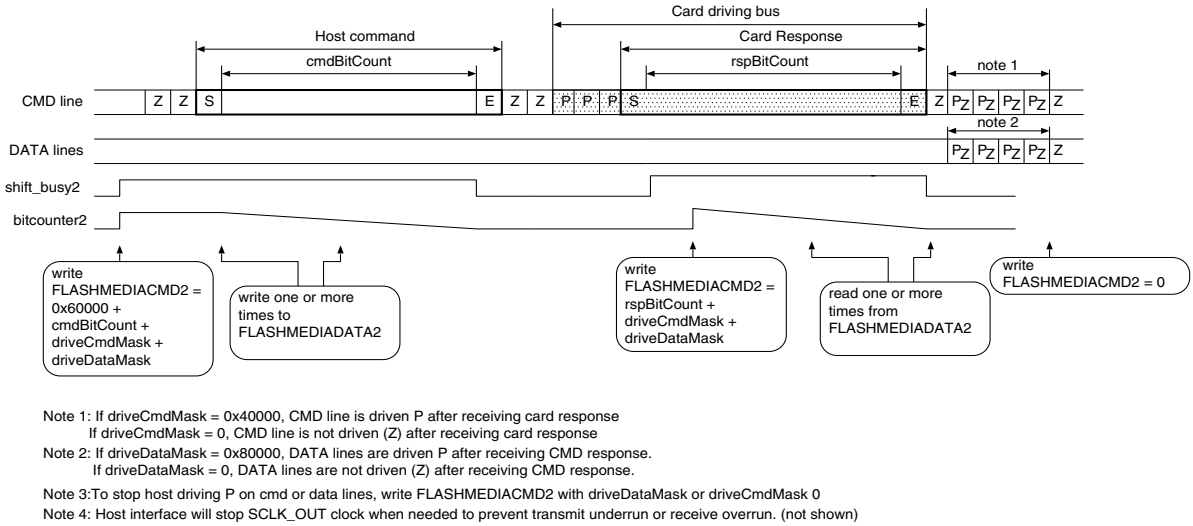
13.5.6 Flash Media Interface Operation in Secure Digital (SD) Mode

All interactions to the Secure Digital (SD) card can be broken down into a number of cascaded elementary operations. There are three elementary operations in SD mode:

- Send command to card

- Read data from card (one or more packets)
- Write data to card (one or more packets)

13.5.6.1 Send Command to Card



Note 1: If driveCmdMask = 0x40000, CMD line is driven P after receiving card response
If driveCmdMask = 0, CMD line is not driven (Z) after receiving card response
Note 2: If driveDataMask = 0x80000, DATA lines are driven P after receiving CMD response.
If driveDataMask = 0, DATA lines are not driven (Z) after receiving CMD response.
Note 3:To stop host driving P on cmd or data lines, write FLASHMEDIACMD2 with driveDataMask or driveCmdMask 0
Note 4: Host interface will stop SCLK_OUT clock when needed to prevent transmit underrun or receive overrun. (not shown)

Figure 13-23. Send Command to Card

The send-command sequence first sends out a command on the CMD line, then receives a card response on the same CMD line. After receiving the card response, the host may drive the CMD and DATA lines depending on the values of the DRIVECMDMASK and DRIVEDATAMASK.

NOTE

Both lines must be driven if the next operation is sending a data packet to the card. The CMD line must be driven, while DATA lines are kept Z when the next operation is receiving data from the card. Both CMD and DATA lines are kept Z when no data follows the command.

While the host is sending data and receiving status from the card, it must look for events on the SHIFTBUSY2 status bit in the FLASHMEDIASTATUS register. It is also possible to capture these events using the SHIFTBUSY2RISE and SHIFTBUSY2FALL interrupts.

To exchange data with the card, the host must write the FLASHMEDIADATA2 register when TX2EMPTY is set, or read FLASHMEDIADATA2 when RCV2FULL is set. This can be done by using interrupts, by polling FLASHMEDIAINTSTAT, or by using a DMA channel on FLASHMEDIADATA2.

A number of bits/bytes/longwords corresponding with CMDBITCOUNT must be written to FLASHMEDIADATA2 during the command transmission. All words, except the first word, contain 32 bits of data. The first word contains the remainder. The data in the first word is left-justified. No CRC logic is present in hardware, so CRC must be inserted by software.

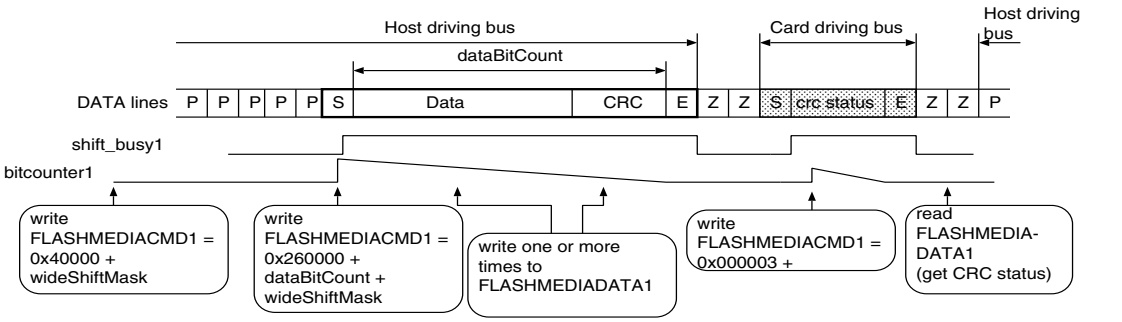
A number of bits/bytes/longwords corresponding with RESPBITCOUNT must be read from FLASHMEDIADATA2 during the response phase. All words, except the first word, contain 32 bits of

data. The first word contains the remainder. The data in the first word is right-justified. No CRC logic is present in hardware, so CRC must be inserted by software.

The writing of RSPBITCOUNT + DRIVECMDMASK + DRIVEDATAMASK to FLASHMEDIACMD2 must take place after SHIFTBUSY2 has gone high.

13.5.6.2 Write Data to Card

The following two timing diagrams show writing data with and without a busy response from the card.

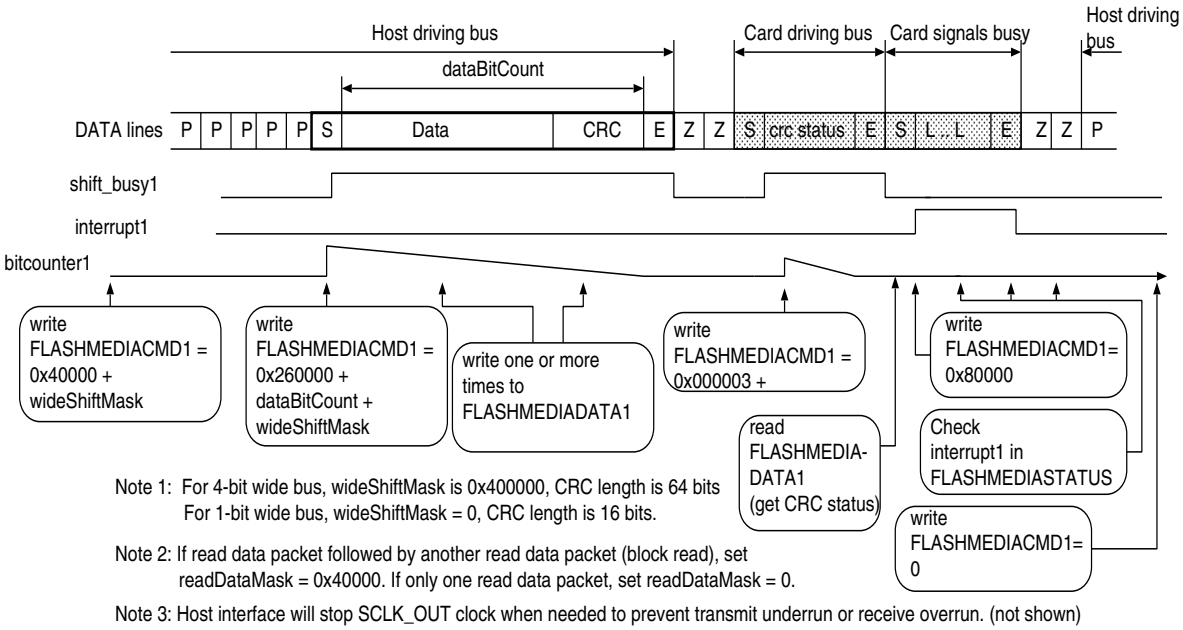


Note 1: For 4-bit wide bus, wideShiftMask is 0x400000, CRC length is 64 bits
For 1-bit wide bus, wideShiftMask = 0, CRC length is 16 bits.

Note 2: If read data packet followed by another read data packet (block read), set readDataMask = 0x40000. If only one read data packet, set readDataMask = 0.

Note 3: Host interface will stop SCLK_OUT clock when needed to prevent transmit underrun or receive overrun. (not shown)

Figure 13-24. Writing to Card With Busy



Note 1: For 4-bit wide bus, wideShiftMask is 0x400000, CRC length is 64 bits
For 1-bit wide bus, wideShiftMask = 0, CRC length is 16 bits.

Note 2: If read data packet followed by another read data packet (block read), set readDataMask = 0x40000. If only one read data packet, set readDataMask = 0.

Note 3: Host interface will stop SCLK_OUT clock when needed to prevent transmit underrun or receive overrun. (not shown)

Figure 13-25. Writing to Card Without Busy

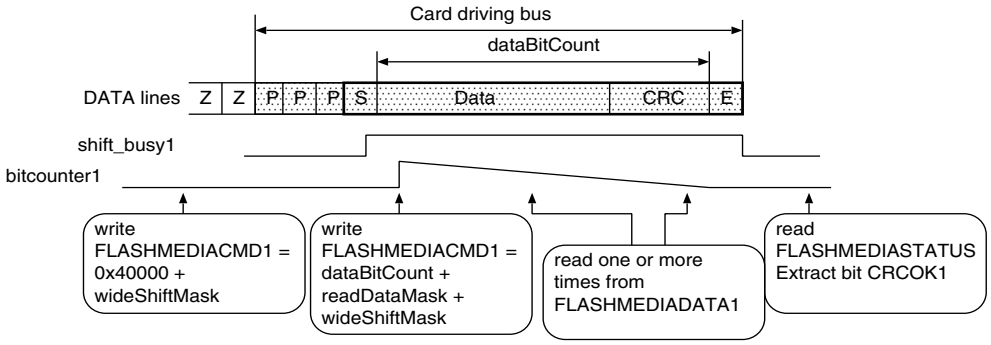
The write sequence sends out a packet on the DATA line, receives a CRC STATUS response from the card, and then looks for a potential busy.

The *DATABITCOUNT* is the number of bits in the packet. This includes the CRC bits. There are 16 CRC bits for each data line (16 CRC bits for the 1-bit bus, 64 CRC bits for the 4-bit bus). The number of bits/bytes/longwords that need to be written to FLASHMEDIADATA1 corresponds with DATABITCOUNT. The user needs to write dummy data instead of the CRC bits to FLASHMEDIADATA1. The CRC value is calculated inside the Flash Media Interface, and the CRC bits written to FLASHMEDIADATA1 are discarded. All words, except the first word, written to FLASHMEDIADATA1 contain 32 bits of data. The first word contains the remainder. Data in the first word must be left-justified.

To read the CRC status, the host must read the FLASHMEDIADATA1 data register once. The CRC status is the three LSB's of the value read.

During this sequence, the host must look for events on SHIFT_BUSY1 and INTERRUPT1. This is accomplished by polling FLASHMEDIASTATUS or FLASHMEDIAINTSTATUS, or by waiting for interrupts SHIFTBUSY1RISE, SHIFTBUSY1FALL, INTERRUPT1RISE, INTERRUPT1FALL.

To read/write data to/from FLASHMEDIADATA1, the host can poll FLASHMEDIAINTSTAT, wait for interrupt, or use a DMA channel. In Figure 13-26, the DATA lines default to a "P" state (strong '1' driven by host). This is only the case if DRIVEDATAMASK was set during last write to FLASHMEDIACMD2. Writing 0x3 to FLASHMEDIACMD1 must take place after SHIFTBUSY1 has gone high. One or more write packets can be sent to the card using this timing diagram.



Note 1: For 4-bit wide bus, wideShiftMask is 0x400000, CRC length is 64 bits
For 1-bit wide bus, wideShiftMask = 0, CRC length is 16 bits.

Note 2: If read data packet followed by another read data packet (block read), set readDataMask = 0x40000. If only one read data packet, set readDataMask = 0.

Note 3: Host interface will stop SCLK_OUT clock when needed to prevent transmit underrun or receive overrun. (not shown)

Figure 13-26. Read Data from Card

The DATABITCOUNT is the number of bits in the packet. This includes the CRC bits. There are 16 CRC bits for the 1-bit bus, 64 CRC bits for the 4-bit bus (16 CRC bits for the 1-bit bus, 64 CRC bits for the 4-bit bus). The number of bits/bytes/longwords that need to be read from FLASHMEDIADATA1 corresponds with DATABITCOUNT.
The CRC can be read from FLASHMEDIADATA1, but the user does need to check the CRC in software. This is done in hardware. The CRC can be checked via bit 0 in register FLASHMEDIASTATUS after packet read. All words, except the first word read from FLASHMEDIADATA1 contain 32 bits of data. The first word contains the remainder. Data in the first word is right-justified.

During this sequence, the host must look for events on SHIFT_BUSY1 and INTERRUPT1. This can be done by polling FLASHMEDIASTATUS or FLASHMEDIAINTSTATUS, or by waiting for interrupts SHIFTBUSY1RISE, SHIFTBUSY1FALL, INTERRUPT1RISE, INTERRUPT1FALL. To read/write data to/from FLASHMEDIADATA1, the host can poll FLASHMEDIAINTSTAT, wait for interrupt, or use a DMA channel. The writing of DATABITCOUNT + READDATAMASK + WIDESHIFTMASK to FLASHMEDIACMD1 must take place after SHIFTBUSY1 has gone high. One or more packets can be received from the card using Figure 13-26.

13.5.7 Commonly Used Commands in SD Mode

Some pseudo-code descriptions are given in this section for send command, read multiple block, and write multiple block commands.

13.5.7.1 Send Command to Card (No Data)

This sequence is intended for commands that require status response from the card, but no data transfer between host and card. There is provision to do CRC insertion or check for command and response packets. All need to be done in software.

```
/* write command to host */
CMDBITCOUNT = 46
FLASHMEDIACMD2 = 0x060000 + CMDBITCOUNT
while(CMDBITCOUNT > 0)
        {
        if(FLASHMEDIADATA2 empty)
                {
                write data to FLASHMEDIADATA2
                CMDBITCOUNT:= CMDBITCOUNT - 32;
                }
        }
/* one of the two waits need to be done. */
/* First one is more suitable for polling */
/* second one more suitable for interrupt driven */
wait until ((FLASHMEDIACMD2 & 0xFFFF) == 0) OR
wait until (SHIFTBUSY2FALL event)

/* receive status from host */
wait until (SHIFTBUSY2RISE event) OR
wait until ((FLASHMEDIASTATUS & 8)!= 0)
RESPBITCOUNT = 46 or 134 /* depends on command */
FLASHMEDIACMD2 = RESPBITCOUNT;
while(RESPBITCOUNT > 0)
        {
        if(FLASHMEDIADATA2 full)
                {
                read data from FLASHMEDIADATA2
                RESPBITCOUNT:= RESPBITCOUNT - 32;
                }
        }
```

13.5.7.2 Send Command to Card (Receive Multiple Data Blocks and Status)

This sequence sends a read data command to the card. The card sends back a response token on the CMD line, while at the same time sending the data on the DATA lines. The sequence is set to receive BLOCKCOUNT data packets from the card. No STOP command is sent as part of this sequence.

```
CMDBITCOUNT = 46
if(wide_shift_mode)
        wide_shift_mask = 0x400000;
else
        wide_shift_mask = 0;
FLASHMEDIACMD2 = 0x460000 + CMDBITCOUNT
FLASHMEDIACMD1 = 0x040000 + wide_shift_mask
while(CMDBITCOUNT > 0)
        {
        if(FLASHMEDIADATA2 empty)
                {
                read FLASHMEDIADATA2
                CMDBITCOUNT = CMDBITCOUNT - 32;
                }
        }
wait until ((FLASHMEDIACMD2 & 0xFFFF) == 0) OR
wait until (SHIFTBUSY2FALL event)
/* start receiving data and status */
RESPBITCOUNT = 46 or 134;
BLOCKCOUNT = <N>;
while(BLOCKCOUNT > 0)
        {
        -- start reception of new block
        DATABITCOUNT = <blocklen> + crclen;
        while(DATABITCOUNT > 0 || RESPBITCOUNT > 0)
                {
                if(RESPBITCOUNT > 0 && SHIFTBUSY2RISE event)
                FLASHMEDIACMD2 = 0x400000 + RESPBITCOUNT;
                if(SHIFTBUSY1RISE event)
                        if(BLOCKCOUNT == 1) /* last block */
                                FLASHMEDIACMD1 = 0x000000 + dataBitCount + wide_shift_mask;
                        else
                                FLASHMEDIACMD1 = 0x040000 + dataBitCount + wide_shift_mask;
                if(FLASHMEDIADATA2 full)
                        {
                        read FLASHMEDIADATA2
                        RESPBITCOUNT = RESPBITCOUNT - 32;
                        }
                if(FLASHMEDIADATA1 full)
                        {
                        read FLASHMEDIADATA1
                        dataBitCount = dataBitCount - 32;
                        }
                }
        if((FLASHMEDIASTATUS & 1) == 1)
                CRC OK!.
        BLOCKCOUNT = BLOCKCOUNT - 1;
}
```

13.5.7.3 Send Command to Card (Write Multiple Data Blocks)

This sequence sends a write data command to the card. The card sends back a response token on the CMD line. After receiving this response, the host starts transmitting data on the DAT lines. After every data packet, the card sends back a CRC status response, followed by a possible busy.

The sequence is set to send BLOCKCOUNT data packets to the card. No STOP command is sent as part of this sequence.

```
/* write command to host */
CMDBITCOUNT = 46
if(wide_shift_mode)
        wide_shift_mask = 0x400000;
else
        wide_shift_mask = 0;
FLASHMEDIACMD2 = 0xC60000 + CMDBITCOUNT
while(CMDBITCOUNT > 0)
        {
        if(FLASHMEDIADATA2 empty)
                {
                write data to FLASHMEDIADATA2
                CMDBITCOUNT:= CMDBITCOUNT - 32;
                }
        }
/* one of the two waits need to be done. */
/* First one is more suitable for polling */
/* second one more suitable for interrupt driven */
wait until ((FLASHMEDIACMD2 & 0xFFFF) == 0) OR
wait until (SHIFTBUSY2FALL event)

/* receive status from host */
wait until (SHIFTBUSY2RISE event) OR
wait until ((FLASHMEDIASTATUS & 8)!= 0)
RESPBITCOUNT = 46 or 134 /* depends on command */
FLASHMEDIACMD2 = 0xC00000 + RESPBITCOUNT;
while(RESPBITCOUNT > 0)
        {
        if(FLASHMEDIADATA2 full)
                {
                read data from FLASHMEDIADATA2
                RESPBITCOUNT:= RESPBITCOUNT - 32;
                }
        }
}
/* start sending data to card */
BLOCKCOUNT:= <N>
while(BLOCKCOUNT > 0)
        {
        -- start transmission of new block
        DATABITCOUNT = <blockLen> + crcLen;
        FLASHMEDIACMD1 = 0x260000 + dataBitCount +
                                wide_shift_mask;
        while(DATABITCOUNT > 0)
                {
                if(FLASHMEDIADATA1 empty)
                        {
                        write data to FLASHMEDIADATA1
```

```
                        DATABITCOUNT = DATABITCOUNT - 32;
                        }
                }
        wait until((FLASHMEDIACMD1 & 0xFFFF) == 0) OR
        wait until (SHIFTBUSY1FALL event)
        -- receive CRC status from card
        wait until (SHIFTBUSY1RISE event) OR
        wait until ((FLASHMEDIASTATUS & 2)!= 0)
        FLASHMEDIACMD1 = 3;
        wait until(FLASHMEDIADATA1 full)
        CRC status = 0x7 & FLASHMEDIADATA1

        FLASHMEDIACMD1 = 0x80000;
                /* wait for interrupt now.
                On rising edge of busy, INTLEVEL1RISE event will
                occur. On falling edge of busy, INTLEVEL1FALL event
                will occur. During busy, (FLASHMEDIASTATUS & 4) == 4
                */
        wait until ((FLASHMEDIASTATUS & 4) == 0) /* busy end */
        FLASHMEDIACMD1 = 0;
        BLOCKCOUNT:= BLOCKCOUNT - 1;
        }
FLASHMEDIACMD2 = 0;
```

Chapter 14
DMA Controller

The direct memory access controller (DMAC) of the MCF5251 quickly and efficiently moves blocks of data with minimal processor overhead. The DMA module, shown in Figure 14-1, provides four channels that allow byte, word, or longword data transfers. These transfers are dual address to on-chip devices; such as the ATA, UART, SDRAM controller, and audio module.

NOTE

DMA transfers to and from the IDE interface are considered memory to memory transfers and therefore there are no specific channel allocations mentioned in this section.

14.1 DMA Features

The features of the DMA controller as follows:

- Four fully independent programmable DMA controller module channels
- Auto-alignment feature for source or destination accesses
- Dual-address transfer capability
- Channels 0 and 1 request signals may be connected to the audio block
- Channels 2 and 3 request signals may be connected to the interrupt lines of UART0 and UART1, respectively
- Any of the four channels request signals may be connected to the ATA module
- Channel arbitration on transfer boundaries
- Data transfers in 8-, 16-, 32-, or 128-bit blocks using a 16-byte buffer
- Burst and cycle steal transfers
- Independent transfer widths for source and destination
- Independent source and destination address registers
- Data transfer in two clocks

14.2 DMA Signal Description

This section contains a brief description of the DMA module signals that provide handshake control for either a source or destination external device. Table 14-1 summarizes these handshake signals.

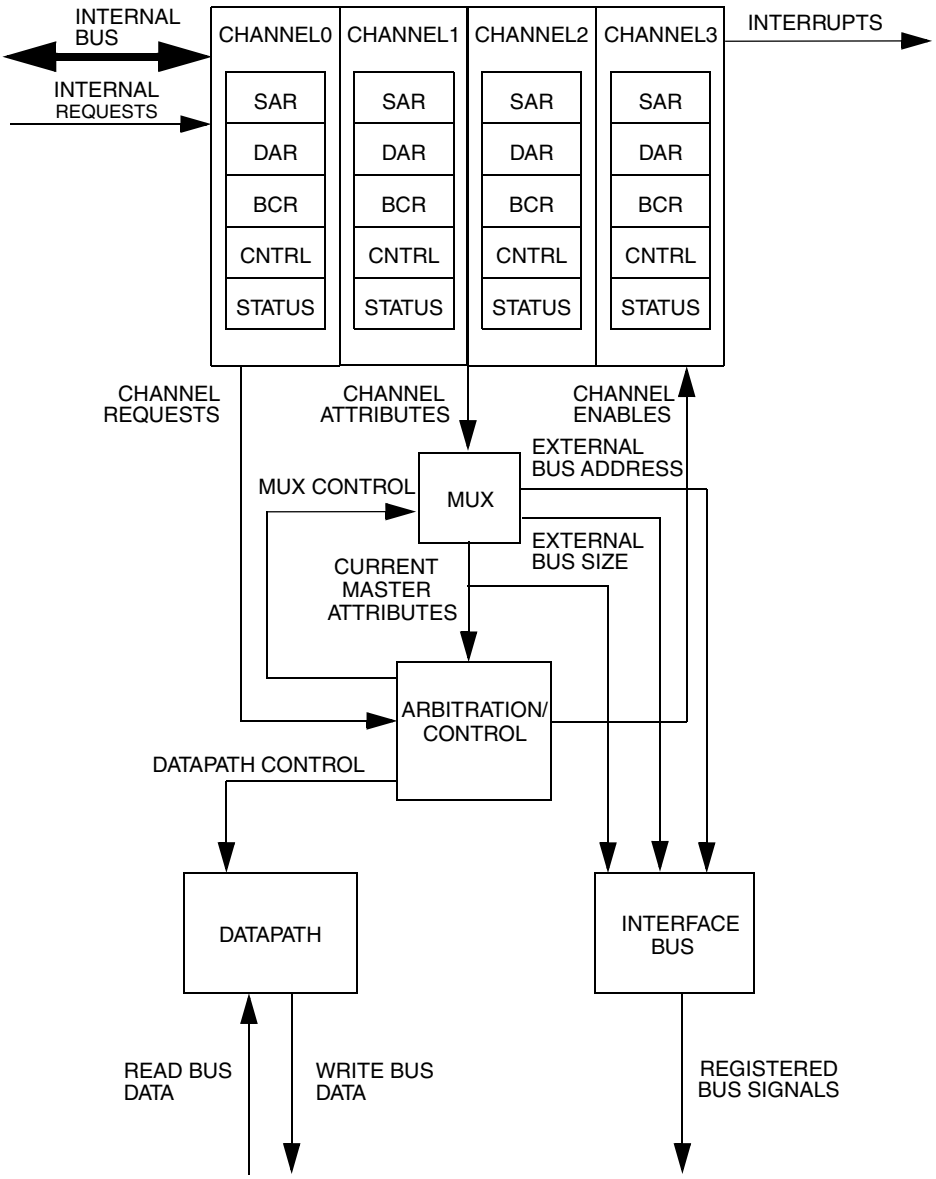


Figure 14-1. DMA Signal Diagram

14.2.1 DMA Request

These internal request signals are DMA inputs. There is one input for each of the four DMA channels. The request sources are selectable by programming the DMAROUTE register. Each DMA channel is programmable individually.

The internal signals are asserted by a peripheral device to request an operand transfer between that peripheral and memory.

14.3 DMA Module Overview

The DMA controller module usually transfers data at rates much faster than the ColdFire core under software control can handle. The term DMA refers to the ability for a peripheral device to access memory in a system in the same manner as the core. DMA operations can greatly increase overall system performance.

The DMA module consists of four independent channels. The term DMA is used throughout this section to reference any of the four channels, as they are all functionally equivalent. It is impossible to implicitly address all four DMA channels at the same time.

DMA requests can be generated by the processor writing to the START bit in the DMA control register or generated by an on-chip peripheral device asserting one of the REQUEST signals. The processor can program the amount of bus bandwidth allocated for the DMA for each channel. The DMA channels support continuous and cycle-steal transfer modes.

The DMA controller supports dual-address transfers. In dual-address mode, the DMA channel supports 32 bits of address and 32 bits of data. Dual-address transfers can be initiated by either a processor request using the START bit or by an internal peripheral device using the REQUEST signal. Two bus transfers occur in this mode, a read from a source device and a write to a destination device (see Figure 14-2).

Any operation involving the DMA module follows the same three basic steps:

1. Channel initialization step—The DMA channel registers are loaded with control information, address pointers, and a byte transfer count. Also, the DMAROUTE register is programmed to control the source of the internal requests.
2. Data transfer step—The DMA accepts requests for operand transfers and provides addressing and bus control for the transfers.
3. Channel termination step—This occurs after operation is complete. The channel indicates the status of the operation in the channel status register.

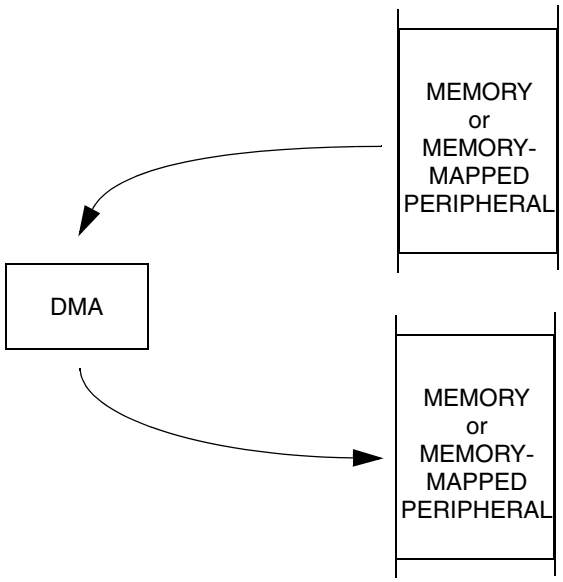


Figure 14-2. Dual Address Transfer

14.4 DMA Memory Map and Register Definitions

The registers of each channel are mapped into memory as shown in Table 14-1.

The DMA control module registers determine the operation of the DMA controller module. This section describes each of the internal registers and its bit assignment.

NOTE

There is no mechanism for preventing a write to a control register during DMA transfer. This situation should be avoided.

Table 14-1. Memory Map DMA Channels

DMA Channel	Address	[31:24]	[23:16]	[15:8]	[7:0]
–	MBAR2 + $188	DMAROUTE - Request source control			

14.4.1 REQUEST Source Selection

The routing control register (DMAroute) controls where the non-processor DMA request for the four DMA channels is coming from.

Address MBAR2 + $188 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	DMA3REQ								DMA2REQ							
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	DMA1REQ								DMA0REQ							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-3. DMAroute Register

Table 14-3. DMAroute Register Field Descriptions

DMAroute Bits	Field Name	DMA Channel
31–24	DMA3REQ (7:0)	DMA3
23–16	DMA2REQ (7:0)	DMA2
15–8	DMA1REQ (7:0)	DMA1
7–0	DMA0REQ (7:0)	DMA0

Table 14-4. DMA3REQ Field Definition

DMA3req (7:0) Field Value	Request Source for DMA3	Block
0x00	DMA3: ATA	ATA
0x80	DMA3: UART1	UART1

Table 14-5. DMA2REQ Field Definition

DMA2req (7:0) Field Value	Request Source for DMA2	Block
0x00	DMA2: ATA	ATA
0x80	DMA2: UART0	UART0

Table 14-6. DMA1REQ Field Definition

DMA1req (7:0) Field Value	Request Source for DMA1	Block
0x00	DMA1: ATA	ATA
0x80	DMA1: audio source 1	audio
0x81	DMA1: audio source 2	audio

Table 14-7. DMA0REQ Field Definition

DMA0req (7:0) Field Value	Request Source for DMA0	Block
0x00	DMA0: ATA	ATA
0x80	DMA0: audio source 1	audio
0x81	DMA0: audio source 2	audio

14.4.2 Source Address Register

The source address register (SAR) is a 32-bit register containing the address from which the DMA controller module requests data during a transfer.

Address MBAR + $300
MBAR+ $340
MBAR + $380
MBAR + $3C0

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	SAR31	SAR30	SAR29	SAR28	SAR27	SAR26	SAR25	SAR24	SAR23	SAR22	SAR21	SAR20	SAR19	SAR18	SAR17	SAR16
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	SAR15	SAR14	SAR13	SAR12	SAR11	SAR10	SAR9	SAR8	SAR7	SAR6	SAR5	SAR4	SAR3	SAR2	SAR1	SAR0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-4. Source Address Register (SAR)

NOTE

Only part of the on-chip SRAM can be accessed by the DMA. The memory controlled by RAMBAR0 is not visible for DMA. The memory controlled by RAMBAR1 is visible for DMA. As a result, the SAR or DAR address range cannot be programmed to on-chip SRAM0 memory, since the on-chip DMAs cannot access on-chip SRAM0 as a source or destination. They can access SRAM1, however.

14.4.3 Destination Address Register

The destination address register (DAR) is a 32-bit register containing the address to which the DMA controller module sends data during a transfer.

Address MBAR + $304
MBAR + $344
MBAR + $384
MBAR + $3C4

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	DAR31	DAR30	DAR29	DAR28	DAR27	DAR26	DAR25	DAR24	DAR23	DAR22	DAR21	DAR20	DAR19	DAR18	DAR17	DAR16
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	DAR15	DAR14	DAR13	DAR12	DAR11	DAR10	DAR9	DAR8	DAR7	DAR6	DAR5	DAR4	DAR3	DAR2	DAR1	DAR0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-5. Destination Address Register (DAR)

NOTE

The MCF5251 on-chip DMAs must be careful when transferring data to cacheable memory since the on-chip DMAs do not maintain cache coherency with the MCF5251 instruction cache.

14.4.4 Byte Count Register

The byte count register (BCR) is a 24-bit register containing the number of bytes remaining to be transferred for a given block. The offset within the memory map is based on the value of the BCR24BIT bit in the MPARK register of the SIM module. See Table 14-6 for the bit locations.

NOTE

If the BCR24BIT = 1, the upper 8 bits are loaded with zeros.

The BCR decrements on the successful completion of the address phase of a write transfer in dual-address mode. The amount the BCR decrements is 1, 2, 4, or 16 for byte, word, longword, or line accesses, respectively.

Address MBAR + $30C
MBAR + $34C
MBAR + $38C
MBAR + $3CC

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R									BCR23	BCR22	BCR21	BCR20	BCR19	BCR18	BCR17	BCR16
W																
Reset	–	–	–	–	–	–	–	–	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	BCR15	BCR14	BCR13	BCR12	BCR11	BCR10	BCR9	BCR8	BCR7	BCR6	BCR5	BCR4	BCR3	BCR2	BCR1	BCR0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-6. Byte Count Register (BCR)—BCR24BIT = 1

Address MBAR + $30C
MBAR + $34C
MBAR + $38C
MBAR + $3CC

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	BCR15	BCR14	BCR13	BCR12	BCR11	BCR10	BCR9	BCR8	BCR7	BCR6	BCR5	BCR4	BCR3	BCR2	BCR1	BCR0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

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

Figure 14-7. Byte Count Register (BCR)—BCR24BIT = 0

The DONE bit in the DMA status register (Table 14-9) is set when the entire block transfer is complete.

When a transfer sequence is initiated and the BCR contains a value that is not divisible by 16, 4, or 2 when the DMA is configured for line, longword, or word transfers, respectively, the configuration error bit (CE) in the DMA status register (DSR) is set and the transfer does not take place. Refer to Section 14.4.6, "DMA Status Register," for more details.

14.4.5 DMA Control Register

The DMA control register (DCR) sets the configuration of the DMA controller module. Depending on the state of the BCR24BIT in the MPARK register in the SIM module, the DMA control register looks slightly different. Specifically, the AT bit (DCR[15]) is included when BCR24BIT = 1, providing greater flexibility in DMA transfer acknowledge.

Address MBAR + $308
MBAR + $348
MBAR + $388
MBAR + $3C8

Access: User read/write

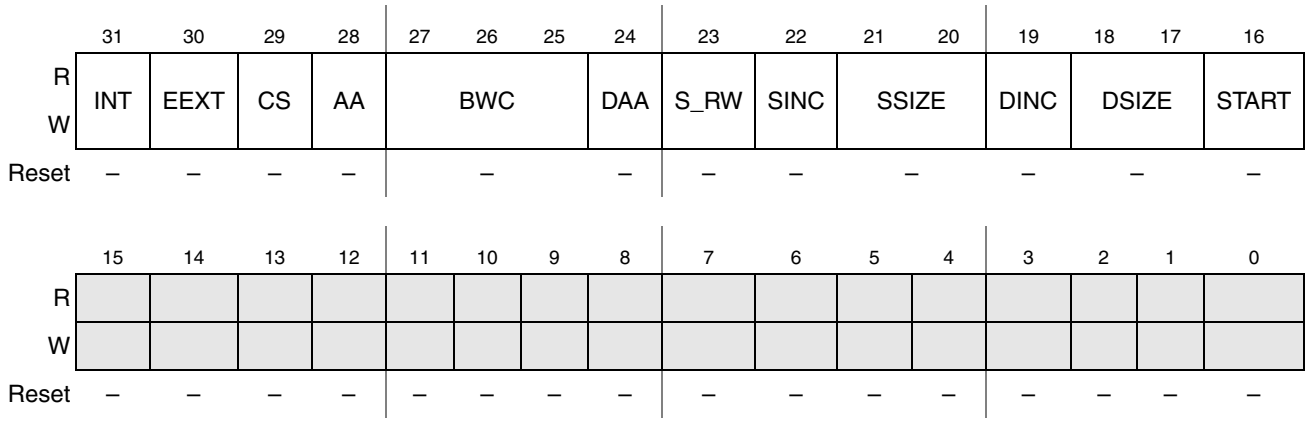


Figure 14-8. DMA Control Register (DCR)—BCR24BIT = 0

Table 14-8. DMA Control Register (DCR) Field Descriptions

Field	Description
31 INT	The Interrupt on completion of transfer field determines whether an interrupt is generated at the completion of the transfer or occurrence of an error condition. 0 No interrupt is generated. 1 = Internal interrupt is enabled.
30 EEXT	Enable peripheral request. Collision could occur between the START bit and the REQUEST signal when EEXT = 1. Therefore, caution should be exercised when initiating a DMA transfer with the START bit while EEXT = 1. 0 Peripheral request is ignored. 1 Enables peripheral request to initiate transfer. Internal request is always enabled. It is initiated by writing a 1 to the START bit.
29 CS	Cycle steal. 0 DMA continuous make read/write transfers until the BCR decrements to zero. 1 Forces a single read/write transfer per request. The request may be processor by setting the START bit, or periphery by asserting the REQUEST signal (Can be generated by the processor).
28 AA	The auto-align bit and the SIZE bits determine whether the source or destination is auto-aligned. Auto alignment means that the accesses are optimized based on the address value and the programmed size. For more information, see Section 14.7.2.2, “Auto Alignment.” 0 Auto-align disabled. 1 If the SSIZE bits indicate a larger or equivalent transfer size with respect to DSIZE, then the source accesses are auto-aligned. If the DSIZE bits indicate a larger transfer size than SSIZE, then the destination accesses are auto-aligned. Source alignment takes precedence over destination alignment. If auto- alignment is enabled, the appropriate address register increments, regardless of the state of DINC or SINC.

Table 14-8. DMA Control Register (DCR) Field Descriptions (continued)

Field	Description
27–25 BWC	The three bandwidth control bits are decoded for internal bandwidth control. When the byte count reaches any multiple of the programmed BWC boundary, the request signal to the internal arbiteris negated until data access completes. This enables the arbiter to give another device access to the bus.Table 14-9 shows the encoding for these bits. When the bits are cleared, the DMA does not negate its request. The 000 encoding asserts a priority signal when the channel is active, signaling that the transfer has been programmed for a higher priority. When the BCR reaches a multiple of the values shown in Table 14-9, the bus is relinquished. For example, if BWC = 001 (512 bytes or value of 0x0200), BCR24BIT = 0, and the BCR is set to 0x1000, the bus is relinquished after BCR values of 0x2000, 0x1E00, 0x1C00, 0x1A00, 0x1800, 0x1600, 0x1400, 0x1200, 0x1000, 0x0E00, 0x0C00, 0x0A00, 0x0800, 0x0600, 0x0400, and 0x0200. In another example, BWC = 110, BCR24BIT = 0, and the BCR is set to 33000. The bus is relinquished after transferring 232 bytes, because the BCR is at 32768, which is a multiple of 16384.
24 DAA	Dual address access. 0 The DMA channel is in dual-address mode. 1 Reserved.
23 S_RW	Reserved, must be set to 0.
22 SINC	The source increment bit determines whether the source address increments after each successful transfer. 0 No change to the SAR after a successful transfer. 1 The SAR increments by 1, 2, 4, or 16; depending upon the size of the transfer.
21–20 SSIZE	The source size field determines the data size of the source bus cycle for the DMA control module. Table 14-10 shows the encoding for this field.
19 DINC	The destination increment bit determines whether the destination address increments after each successful transfer. 0 No change to the DAR after a successful transfer. 1 The DAR increments by 1, 2, 4, or 16; depending upon the size of the transfer.

Table 14-9. BWC Encoding

BWC	Block Size	
	BCR24BIT = 0	BCR24BIT = 1
000	DMA has priority	
001	512	16384
010	1024	32768
011	2048	65536
100	4096	131072
101	8192	262144
110	16384	524288
111	32768	1048576

Table 14-10. SSIZE Encoding

SSize	Transfer Size
00	Longword
01	Byte
10	Word
11	Line

Table 14-8. DMA Control Register (DCR) Field Descriptions (continued)

Field	Description
18–17 DSIZE	The Destination Size field determines the data size of the destination bus cycle for the DMA controller module. Table 14-11 shows the encoding for this field. **Table 14-11. DSIZE Encoding** SSize / Transfer Size: 00 Longword; 01 Byte; 10 Word; 11 Line
16 START	Start transfer. 0 DMA inactive. 1 The DMA begins the transfer in accordance to the values in the control registers. This bit is self-clearing after one clock and is always read as logic 0.
15–0	Reserved.

14.4.6 DMA Status Register

The 8-bit DMA status register (DSR) indicates the status of the DMA controller module. The DMA controller module, in response to an event, writes to the appropriate bit in the DSR. Only a write to the DONE bit (DSR[0]) results in action. Setting the DONE bit creates a single-cycle pulse which resets the channel, thus clearing all bits in the register. The DONE bit is set at the completion of a transfer or during the transfer to abort the access.

Table 14-9 shows the detailed structure of the DMA status register.

Address MBAR + $310
MBAR + $350
MBAR + $390
MBAR + $3D0 — Access: User read/write

	7	6	5	4	3	2	1	0
R		CE	BES	BED		REQ	BSY	DONE
W								
Reset	–	0	0	0	–	1	1	1

Figure 14-9. DMA Status Register (DSR)

Table 14-12. DMA Status Register (DSR) Field Descriptions

Field	Description
Bit 7	Reserved
6 CE	A configuration error results when either the number of bytes represented by the BCR is not consistent with the requested source or destination transfer size. Configuration error can also result from the SAR or DAR containing an address that does not match the requested transfer size for the source or destination. The bit is cleared during a hardware reset or by writing a one to DSR[DONE]. 0 No configuration error exists. 1 A configuration error has occurred.
5 BES	Bus error on source. 0 No bus error occurred. 1 The DMA channel terminated with a bus error either during the read portion of a transfer.
4 BED	Bus error on destination. 0 No bus error occurred. 1 The DMA channel terminated with a bus error during the write portion of a transfer.
3	Reserved
2 REQ	Request 0 There is no request pending or the channel is currently active. The bit is cleared when the channel is selected. 1 The DMA channel has a transfer remaining and the channel is not selected.
1 BSY	Busy 0 DMA channel is inactive. This bit is cleared when the DMA has finished the last transaction. 1 This bit is set the first time the channel is enabled after a transfer is initiated.
0 DONE	The transaction done bit may be read or written and is set when all DMA controller module transactions have completed normally, as determined by the transfer count and error conditions. When the BCR reaches zero, DONE is set at the successful conclusion of the final transfer. Writing a 1 to this bit clears all DMA status bits and therefore can be used as an interrupt handler to clear the DMA interrupt and error bits. The DONE bit can also be used to abort a transfer in progress by resetting the status bits. The DONE bit is self clearing. Therefore, writing a 0 to it has no effect. 0 Writing or reading a 0 has no effect whatsoever. 1 DMA transfer completed.

14.4.7 DMA Interrupt Vector Register

The DMA Interrupt Vector Register (DIVR) is an 8-bit register, which is driven out onto the bus in response to an internal acknowledge cycle.

Address MBAR + $314
MBAR + $354
MBAR + $394
MBAR + $3D4 — Access: User read/write

	7	6	5	4	3	2	1	0
R W	INTERRUPT VECTOR BITS							
Reset	0	0	0	0	1	1	1	1

Figure 14-10. DMA Interrupt Vector Register (DIVR)

14.5 Transfer Request Generation

The DMA channel supports processor and periphery requests. Bus utilization can be minimized for either processor or periphery request by selecting between cycle-steal and continuous modes. The DCR[EEXT] field determines the request-generation method for each channel.

14.5.1 Cycle-Steal Mode

The DMA is in cycle-steal mode if the CS field (DCR[29]) is set. In this mode, only one complete transfer from source to destination takes place for each request. Depending on the state of the EEXT field (DCR[30]), the request can be either processor or periphery. Processor request is selected by setting the START bit (DCR[16}). Periphery request is initiated by asserting the REQUEST signal while the EEXT bit is set.

14.5.2 Continuous Mode

The DMA is in continuous mode If the CS field (DCR[29]) is cleared. After an internal or external request is asserted, the DMA continuously transfers data until the byte count register (BCR) reaches zero or the DONE bit (DSR[0]) is set.

The continuous mode can operate at maximum or limited rate. The maximum rate of transfer can be achieved if the bandwidth control BWC field (DCR[27:25]) is programmed to 000. Then the active DMA channel continues until the BCR decrements to zero or the DONE bit is set.

A limited rate can be achieved by programming the BWC field to any value other than 000. The DMA performs the specified number of transfers, then relinquishes control of the bus. The DMA negates its internal bus request on the last transfer before the BCR reaches a multiple of the boundary specified in the BWC field. On transfer completion, the DMA asserts its bus request again to regain bus ownership at the earliest opportunity, as determined by the internal bus arbiter. The minimum time that the DMA loses bus control is one bus cycle.

14.6 Data Transfer Modes

Each DMA channel supports dual-address transfers. The dual-address transfer mode consists of a source operand read and a destination operand write.

14.6.1 Dual-Address Transaction

The DMA controller module begins a dual-address transfer sequence when the DAA bit (DCR[24]) is cleared during a DMA request. If no error condition exists, the REQ bit (DSR[2]) is set.

14.6.1.1 Dual-Address Read

The DMA controller module will drive the value in the source address register (SAR) onto the internal address bus. If the SINC bit (DCR[22]) is set, then the SAR increments by the appropriate number of bytes upon a successful read cycle. When the appropriate number of read cycles completes successfully, the DMA initiates the write portion of the transfer.

In the event of a termination error, the BES (DSR[5]) and DONE bit (DSR[0]) are set and no further DMA transactions take place.

14.6.1.2 Dual-Address Write

The DMA controller module drives the value in the destination address register (DAR) onto the address bus. If the DINC bit (DCR[19]) is set, then the DAR increments by the appropriate number of bytes at the completion of a successful write cycle. The byte count register (BCR) decrements by the appropriate number of bytes. The DONE bit (DSR[0]) is set when the BCR reaches zero. If the BCR is greater than zero, then another read/write transfer is initiated. If the byte count register (BCR) is a multiple of the programmed bandwidth control (BWC), then the DMA request signal is negated until termination of the bus cycle to allow the internal arbiter to switch masters.

In the event of a termination error, the BES (DSR[5]) and DONE bit (DSR[0]) are set and no further DMA transactions takes place.

14.7 DMA Transfer Functional Description

In the following section, the term DMA request implies that the START bit (DCR[16]) is set or the EEXT bit (DCR[30]) is set, followed by assertion of REQUEST. The START bit is cleared when the channel begins an internal access.

Before initiating a transfer, the DMA controller module verifies that the source size (SSIZE = DSC[21:20]) and destination size (DSIZE = DSR[18:17]) for dual-address access are consistent with the source address and destination address. The CE bit is also set if inconsistency is found between the destination size and the source size in the BCR for dual-address access. If a misalignment is detected, no transfer occurs and the configuration error bit (CE = DSR[6]) is set. Depending on the configuration of the DCR, an interrupt event may be issued when the CE bit is set.

NOTE

If the auto-align bit (AA = DCR[28]) is set, error checking is performed on the appropriate registers only.

A read/write transfer refers to a dual-address access in which a number of bytes are read from the source address and written to the destination address. The number of bytes in the transfer is determined by the larger of the sizes specified by the source and destination size encoding. See Table 14-10 and Table 14-11.

The source and destination address registers (SAR and DAR) increment at the completion of a successful address phase. The BCR decrements at the completion of a successful address write phase. A successful address phase occurs when a valid address request is not held by the arbiter.

14.7.1 Channel Initialization and Startup

Before starting a block transfer operation, the channel registers must be initialized with information describing the channel configuration, request-generation method, and data block. This initialization is accomplished by programming the appropriate information into the channel registers.

14.7.1.1 Channel Prioritization

The four DMA channels are prioritized in ascending order (channel 0 having highest priority and channel 3 having the lowest) or as determined by the BWC bits in the DCR. If the BWC bits for a DMA channel are set to 000, then that channel has priority over the channel immediately preceding it. For example, if DMA channel 3 has the BWC bits set to 000, it has priority over DMA channel 2 but not over DMA channel 1. This is assuming that DMA channel 2 has something other than all zeroes in the BWC bits.

Another example would be the case where the BWC bits in only DMA 2 and DMA 1 are all zeroes. In this case, DMA 1 would have priority over DMA 0 and DMA 2. The BWC bits being zero in DMA 2 in this case have no effect on prioritization.

In the case of simultaneous external requests, the prioritization is either ascending or as determined by each channels BWC bits as described in the previous paragraphs.

14.7.1.2 Programming the DMA

The following are some general comments on programming the DMA:

- No mechanism exists for preventing writes to control registers during DMA accesses
- If the BWC of sequential channels are equivalent, channel priority is in ascending order

The SAR is loaded with the source (read) address. If the transfer is from a peripheral device to memory, the source address is the location of the peripheral data register. If the transfer is from memory to a peripheral device or memory to memory, the source address is the starting address of the data block. This address can be any byte address.

The DAR should contain the destination (write) address. If the transfer is from a peripheral device to memory, or memory to memory, the DAR is loaded with the starting address of the data block to be written. If the transfer is from memory to a peripheral device, the DAR is loaded with the address of the peripheral data register. This address can be any byte address.

The manner in which the SAR and DAR change after each cycle depends on the values in the DCR SSIZE and DSIZE fields and the SINC and DINC bits, and the starting address in the SAR and DAR. If programmed to increment, the increment value is 1, 2, 4, or 16 for byte, word, longword, or line operands, respectively. If the address register is programmed to remain unchanged (no count), the register is not incremented after the operand transfer.

The BCR must be loaded with the number of byte transfers that are to occur. This register is decremented by 1, 2, 4, or 16 at the end of each transfer. The DSR must be cleared for channel startup.

Once the channel has been initialized, it is started by writing a one to the START bit in the DCR or asserting the REQUEST signal, depending on the status of the EEXT bit in the DCR. Programming the channel for processor request causes the channel to request the bus and start transferring data immediately. If the channel is programmed for periphery request, $\overline{\text{REQUEST}}$ must be asserted before the channel requests the bus.

If any fields in the DCR are modified while the channel is active, that change is effective immediately. To avoid any problems with changing the setup for the DMA channel, a 1 should be written to the DONE bit in the DSR to stop the DMA channel.

14.7.2 Data Transfer

14.7.2.1 Periphery Request Operation

All channels can initiate transfers to/from a periphery module by means of REQUEST[3:0]. Source where REQUEST is coming from is programmed in register DMAROUTE. If the EEXT bit (DCR[30]) is set, when a REQUEST is asserted, the DMA initiates a transfer provided the channel is idle. If the CS (cycle steal) bit is set, the read/write transaction on the bus is limited to a single transfer. If the CS bit is clear, multiple read/write transfers can occur on the bus as programmed. REQUEST does not need to be negated until the DONE bit (DSR[0]) is set.

14.7.2.2 Auto Alignment

This feature allows for block transfers to occur at the optimum size based on the address, byte count, and programmed size. To use this feature, AA in the DCR must be set. The source is auto-aligned when the SSIZE bits indicate a larger transfer size compared to DSIZE. Source alignment takes precedence over the destination when the source and destination sizes are equal. Otherwise, the destination is auto-aligned. The address register that is chosen for alignment increments regardless of the value of the increment bit. Configuration error checking is performed on the registers that are not chosen for alignment.

If the BCR contains a value greater than 16, the address will determine the size of the transfer. Single byte, word or longword transfers will occur until the address is aligned to the programmed size boundary, at which time the programmed size accesses begin. When the BCR is less than 16 at the beginning of a read/write transfer, the number of bytes remaining will dictate the transfer size, longword, word or byte.

For example:

AA = 1, SAR = $0001, BCR = $00f0, SSIZE = 00 (longword) and DSIZE = 01 (byte),

Because the SSIZE > DSIZE, the source is auto-aligned. Error checking is performed on the destination registers. The sequence of accesses is as follows:

1. Read byte from $0001—write byte, increment SAR
2. Read word from $0002—write 2 bytes, increment SAR
3. Read long word from $0004—write 4 bytes, increment SAR
4. Repeat longwords until SAR = $00f0
5. Read byte from $00f0—write byte, increment SAR.

If DSIZE is set to another size, then the data writes are optimized to write the largest size allowed based on the address, but not exceeding the configured size.

14.7.2.3 Bandwidth Control

This feature makes provision to force the DMA off the bus to allow another master access. Bus arbiter design was simplified by making arbitration programmable. The decode of the DCR[BWC] field provides 7 levels of block transfer sizes. If the BCR decrements to a value that is a multiple of the decode of the BWC, the DMA bus request negates until termination of the bus cycle. Should a request be pending, the arbiter may then choose to switch the bus to another master. If auto-alignment is enabled (DCR[AA] = 1),

the BCR may skip over the programmed boundary. In this case, the DMA bus request will not be negated. If the BWC = 000, the request signal will remain asserted until the BCR reaches zero. In addition, will assert to indicate that the channel has been programmed to have priority.

NOTE

In this arbitration scheme, the arbiter can always force the DMA to relinquish the bus.

14.7.3 Channel Termination

14.7.3.1 Error Conditions

When the bus encounters a read or write cycle that terminates with an error condition, the appropriate bit of the DSR is set, depending on whether the bus cycle was a read (BES) or a write (BED). The DMA transfers are then halted. If the error condition occurred during a write cycle, any data remaining in the internal holding register is lost.

14.7.3.2 Interrupts

If the INT bit of the DCR is set, the DMA will drive the appropriate slave bus interrupt signal. The processor can then read the DSR to determine if the transfer terminated successfully or with an error. The DONE bit of the DSR is then written with a 1 to clear the interrupt, along with clearing the DONE and error bits.

Chapter 15
UART Modules

This chapter provides signal descriptions, operation, memory map, and register descriptions, an initialization sequence of the three UART modules.

The MCF5251 contains three asynchronous receiver/transmitters (UARTs) that act independently. Each UART is clocked by the system clock. This chapter applies to all three UARTs, which are functionally identical. Refer to Section 15.4, "UART Memory Map and Register Definitions," for addressing differences.

Each UART module, shown in Figure 15-1, consists of the following functional areas:

- Serial Communication Channel
- 16-Bit Baud-Rate Timer
- Internal Channel Control Logic
- Interrupt Control Logic

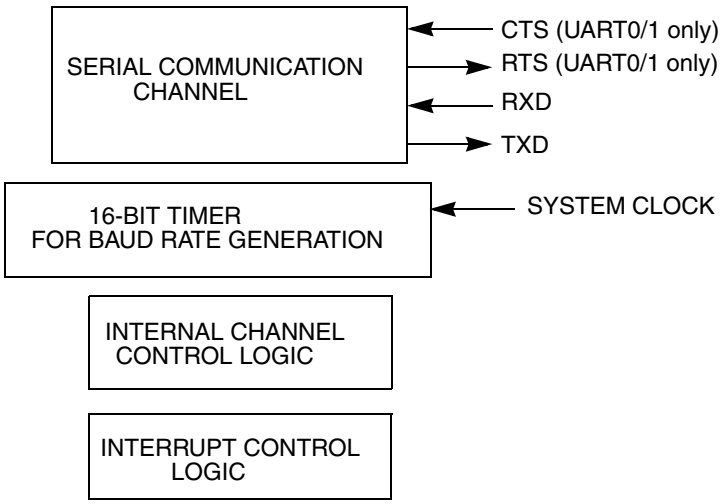


Figure 15-1. UART Block Diagram

15.1 UART Module Features

The MCF5251 contains three independent UART modules. Features of each UART module include the following:

- UART clocked by the system clock
- Full duplex asynchronous receiver/transmitter channel
- Quadruple-buffered receiver
- Double-buffered transmitter

- Independently programmable receiver and transmitter clock source
- Programmable data format
  - Five to eight data bits plus parity
  - Odd, even, no parity, or force parity
  - .563 to 2 stop bits in x16 mode (asynchronous)/1or 2 stop bits in synchronous mode
- Programmable channel modes:
  - Normal (full duplex)
  - Automatic echo
  - Local loopback
  - Remote loopback
- Automatic wakeup mode for multidrop applications
- Four maskable interrupt conditions
- Parity, framing, break, 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

15.1.1 Serial Communication Channel

The communication channel provides a full duplex asynchronous receiver and transmitter using an operating frequency derived from the system clock.

The transmitter accepts parallel data from the CPU; converts it to a serial bit stream; inserts the appropriate start, stop, and optional parity bits; then outputs a composite serial data stream on the channel transmitter serial data output (TxD). Refer to Section 15.3.2.1, “Transmitter,” for additional information.

The receiver accepts serial data on the channel receiver serial data input (RxD); converts it to parallel format; checks for a start bit, stop bit, parity (if any), or any error condition; and transfers the assembled character onto the bus during read operations. The receiver can be polled or interrupt driven. Refer to Section 15.3.2.2, “Receiver,” for additional information.

15.1.2 Baud-Rate Generator/Timer

The 16-bit timer, clocked by the system clock, can function as an asynchronous x16 clock. The baud-rate timer is part of each UART and not related to the ColdFire timer modules.

15.1.3 Interrupt Control Logic

An internal interrupt request signal ($\overline{\text{IRQ}}$) notifies the MCF5251 interrupt controller of an interrupt condition. The output is the logical NOR of all (as many as four) unmasked interrupt status bits in the UART Interrupt Status Register (UISR). The UART Interrupt Mask Register (UIMR) can be programmed to determine which interrupts will be valid in the UISR.

The UART module interrupt level in the MCF5251 interrupt controller is programmed external to the UART module. The UART can be configured to supply the vector from the UART Interrupt Vector Register (UIVR) or the SIM can be programmed to provide an autovector when a UART interrupt is acknowledged.

The interrupt level, priority within the level, and autovectoring capability can also be programmed in the SIM register ICR_U1.

15.2 UART Module Signal Definitions

The following paragraphs contain a brief description of the UART module signals. Figure 15-2 shows both the external and internal signal groups.

NOTE

The terms assertion and negation are used throughout this chapter to avoid confusion when dealing with a mixture of active-low and active-high signals. The term assert or assertion indicates that a signal is active or true, independent of the level represented by a high or low voltage. The term negate or negation indicates that a signal is inactive or false.

15.2.1 Transmitter Serial Data Output

The multiplexed signals TXD0/GPIO45, SCL1/TXD1/GPIO10 and XTRIM/TXD2/GPIO0 can be programmed as general purpose outputs or transmitter serial data outputs. When used as transmitters, the output is held high ('‘mark’' condition) when the transmitter is disabled, idle, or operating in the local loopback mode. Data is shifted out on this signal on the falling edge of the clock source, with the least significant bit transmitted first.

15.2.2 Receiver Serial Data Input

The multiplexed signals RXD0/GPIO46, SDA1/RXD1/GPIO44, and EF/RXD2/GPIO6 can be programmed as general purpose inputs or receiver serial data inputs. When used as receivers, data received on this signal is sampled on the rising edge of the clock source, with the least significant bit received first.

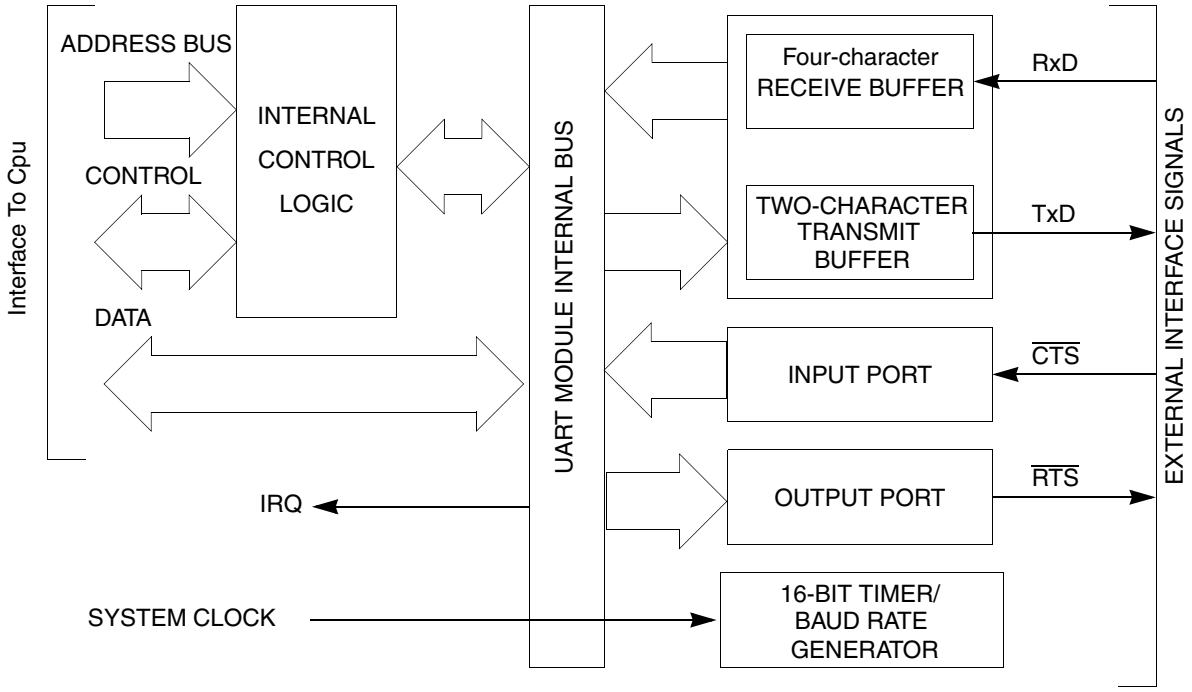


Figure 15-2. External and Internal Interface Signals

15.2.3 Request-To-Send

The Request-To-Send ($\overline{\text{RTS}}$) pins DDATA3/$\overline{\text{RTS0}}$/GPIO4 and DDATA1/$\overline{\text{RTS1}}$/SDATA2_BS2/GPIO2 are multiplexed. When programmed for $\overline{\text{RTS}}$, this active-low output signal can be programmed to be automatically negated and asserted by either the receiver or transmitter. When connected to the clear-to-send ($\overline{\text{CTS}}$) input of a transmitter, this signal controls serial data flow.

UART2 has no RTS signal capability.

15.2.4 Clear-To-Send

The multiplexed signals DDATA2/$\overline{\text{CTS0}}$/GPIO3 and DDATA0/$\overline{\text{CTS1}}$/SDATA0_SDIO1/GPIO1 can be programmed as general purpose inputs or Clear-To-Send inputs. When programmed as ($\overline{\text{CTS}}$), this active-low input is the clear-to-send input and can generate an interrupt on change-of-state.

UART2 has no CTS signal capability.

15.3 Operation

The following sections describe the operation of the baud-rate generator, transmitter and receiver, and other operating modes of the UART module.

15.3.1 Baud-Rate Generator/Timer

The timer references made here relative to clocking the UART are different than the MCF5251 timer module that is integrated on the bus of the ColdFire core. The UART has a baud generator based on an internal baud-rate timer that is dedicated to the UART. The Clock Select Register (UCSR) needs to be programmed to enable the baud-rate timer. With the baud-rate timer, a prescaler supplies an asynchronous 32x clock source to the baud-rate timer. The baud-rate timer register value is programmed with the UBG1 and UBG2 registers. See Section 15.4.12, "Baud Rate Generator (MSB) Register (UBG1n)," for more information.

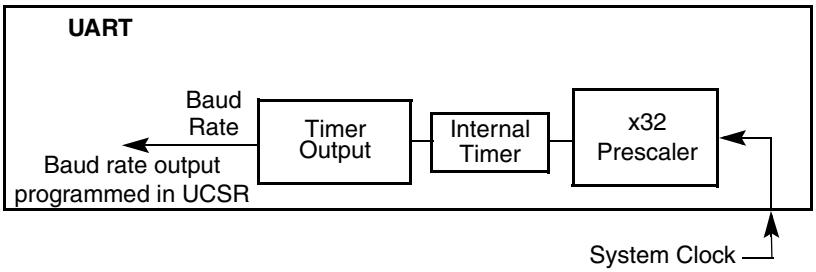


Figure 15-3. Baud-Rate Timer Generator Diagram

15.3.1.1 Calculating Baud Rates

The system clock 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:

$$\text{Baudrate} = \frac{f_{sys}}{[32 \times \text{Divider}]} \qquad \textit{Eqn. 15-1}$$

Using a 80-MHz system clock and letting baud rate equal 19200, then

$$\text{Divider} = \frac{80\text{MHz}}{[32 \times 19200]} = 130(\text{decimal}) = 0\text{x}0082(\text{hexadecimal}) \qquad \textit{Eqn. 15-2}$$

Therefore, UBG1*n* equals 0x00 and UBG2*n* equals 0x82.

15.3.2 Transmitter and Receiver Operating Modes

The functional block diagram of the transmitter and receiver, including command and operating registers, is shown in Figure 15-4. The following paragraphs describe these functions in reference to this diagram. For detailed register information, refer to Section 15.4, "UART Memory Map and Register Definitions."

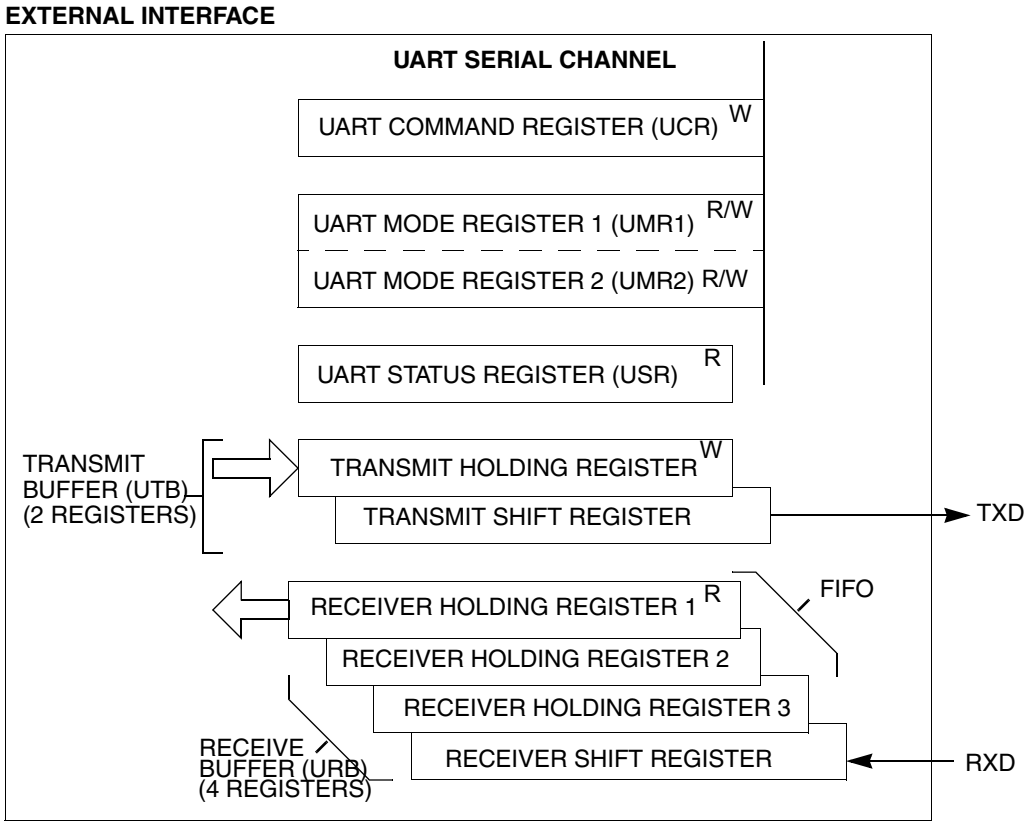


Figure 15-4. Transmitter and Receiver Functional Diagram

15.3.2.1 Transmitter

The transmitter is enabled through the UART command register (UCR) located within the UART module. The UART module signals the CPU when it is ready to accept a character by setting the transmitter-ready bit (TxRDY) in the UART status register (USR). Functional timing information for the transmitter is shown in Figure 15-5.

The transmitter converts parallel data from the CPU to a serial bit stream on TxD. It automatically sends a start bit followed by:

- The programmed number of data bits
- An optional parity bit
- The programmed number of stop bits

The least significant bit is sent first. Data is shifted from the transmitter output on the falling edge of the clock source.

After the transmission of the stop bits, if a new character is not available in the transmitter holding register, the TxD output remains in the high (mark condition) state, and the transmitter-empty bit (TxEMP) in the USR is set. Transmission resumes and the TxEMP bit is cleared when the CPU loads a new character into the UART transmitter buffer (UTB). If the transmitter receives a disable command, it continues operating until the character (if one is present) in the transmit-shift register is completely shifted out of transmitter

TxD. If the transmitter is reset through a software command, operation ceases immediately (refer to Section 15.4.5, “Command Registers (UCRn)”). The transmitter is re-enabled through the UCR to resume operation after a disable or software reset.

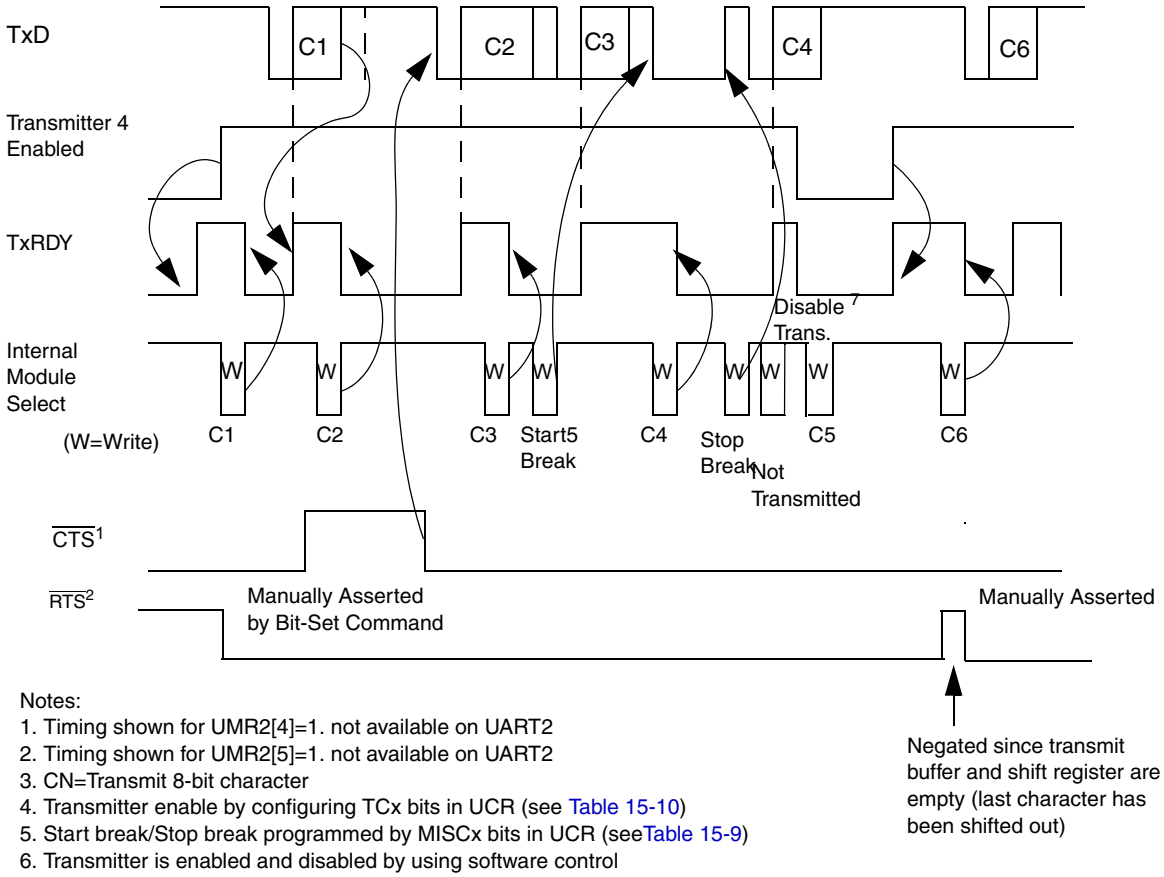


Figure 15-5. Transmitter Timing Diagram

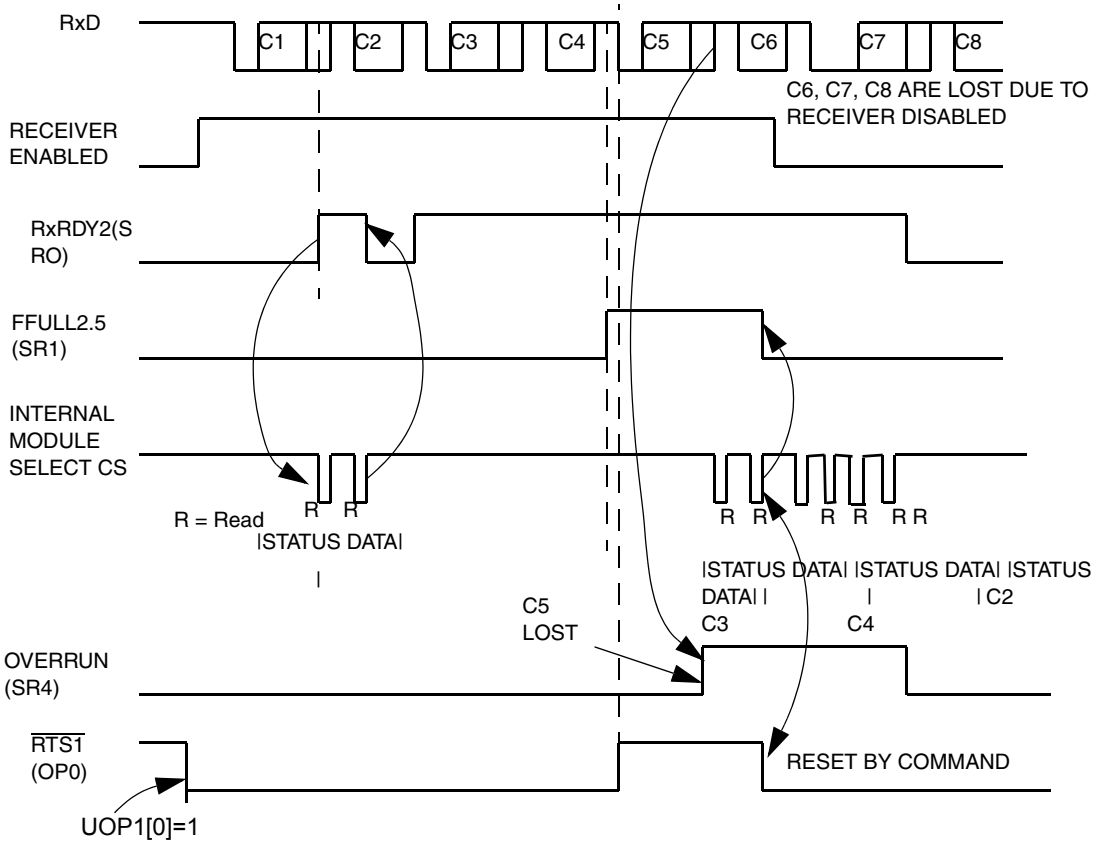
If clear-to-send operation is enabled, $\overline{CTS}$[1] must be asserted for the character to be transmitted. If $\overline{CTS}$ is negated in the middle of a transmission, the character in the shift register is transmitted and following the completion of STOP bits TxD, enters in the mark state until $\overline{CTS}$ is asserted again. If the transmitter is forced to send a continuous low condition by issuing a Send-Break command, the transmitter ignores the state of $\overline{CTS}$.

Users can program the transmitter to automatically negate the request-to-send ($\overline{RTS}$) output on completion of a message transmission. If the transmitter is programmed to operate in this mode, $\overline{RTS}$ must be manually asserted before a message is transmitted. In applications where the transmitter is disabled after transmission is complete and $\overline{RTS}$ is appropriately programmed, $\overline{RTS}$ is negated one bit time after the character in the shift register is completely transmitted. Users must manually enable the transmitter by setting the enable-transmitter bit in the UART Command Register (UCR).

1. CTS and RTS are not available on UART2.

15.3.2.2 Receiver

The receiver is enabled through the UCR located within the UART module. Functional timing information for the receiver is shown in Figure 15-6. The receiver looks for a high-to-low (mark-to-space) transition of the start bit on RxD. When a transition is detected the start bit is validated 0.5 baud clock after the transition. If RxD is sampled high, the start bit is not valid and the search for the valid start bit repeats. If RxD is still low, a valid start bit is assumed and the receiver continues to sample the input at one-bit time intervals at the theoretical center of the bit.



NOTES:
1. TIMING SHOWN FOR UMR1[7]=1
2. TIMING SHOWN FOR UMR1[6]=0
3. CN = RECEIVED 5-8 BIT CHARACTER

Figure 15-6. Receiver Timing Diagram

This process continues until the proper number of data bits and parity (if any) is assembled and one stop bit is detected. Data on the RxD input is sampled on the rising edge of the programmed clock source. The least significant bit is received first. The data is then transferred to a receiver holding register and the RxRDY bit in the USR is set. If the character length is less than eight bits, the most significant unused bits in the receiver holding register are cleared. The Rx RDY bit in the USR is set at the one-half point of the stop bit.

After the stop bit is detected, the receiver immediately looks for the next start bit. However, if a nonzero character is received without a stop bit (framing error) and RxD remains low for one-half of the bit period after the stop bit is sampled, the receiver operates as if a new start bit is detected. The parity error (PE),

framing error (FE), overrun error (OE), and received break (RB) conditions (if any) set error and break flags in the USR at the received character boundary and are valid only when the RxRDY bit in the USR is set.

If a break condition is detected (RxD is low for the entire character including the stop bit), a character of all zeros is loaded into the receiver holding register and the Receive Break (RB) and RxRDY bits in the USR are set. The RxD signal must return to a high condition for at least one-half bit time before a search for the next start bit begins.

The receiver will detect the beginning of a break in the middle of a character if the break persists through the next character time. When the break begins in the middle of a character, the receiver places the damaged character in the receiver first-in-first-out (FIFO) stack and sets the corresponding error conditions and RxRDY bit in the USR. The break persists until the next character time, the receiver places an all-zero character into the receiver FIFO, and sets the corresponding RB and RxRDY bits in the USR. Interrupts can be enabled on receive break.

15.3.2.3 Receiver FIFO

The FIFO is used in the UART receiver 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 RxD (refer to Figure 15-4). Data is assembled in the receiver shift register and loaded into the top empty receiver holding register position of the FIFO. Thus, 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 error-mode bit (ERR) in the channel's mode register (UMR1), status can be provided in character or block modes.

The RxRDY bit in the USR is set whenever one or more characters are available to be read by the CPU. A read of the receiver 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 stack positions are filled with data. Either the RxRDY or FFULL bit can be selected to cause an interrupt.

Character and block modes are two error modes that can be selected within the UMR.

In the character mode, status provided in the USR is given on a character-by-character basis and thus applies only to the character at the top of the FIFO. In the block mode, the status provided in the USR is the logical OR of all characters coming to the top of the FIFO since the last reset error command. A continuous logical OR function of the corresponding status bits is produced in the USR as each character reaches the top of the FIFO.

The block mode is useful in applications where the software overhead of checking each character's error cannot be tolerated. In this mode, entire messages are received and only one data integrity check is performed at the end of the message. This mode has a data-reception speed advantage; however, each character is not individually checked for error conditions by software. If an error occurs within the message, the error is not recognized until the final check is performed, and no indication exists as to which message character is at fault.

In either mode, reading the USR does not affect the FIFO. The FIFO is popped only when the receive buffer is read. The USR should be read prior to reading the receive buffer. If all three of the FIFO receiver holding registers are full when a new character is received, the new character is held in the receiver shift register until a FIFO position is available. If an additional character is received during this state, the contents of the FIFO are not affected. However, the previous character in the receiver shift register is lost and the OE bit in the USR is set when the receiver detects the start bit of the new overrunning character.

To support flow control capability, the receiver can be programmed to automatically negate and assert $\overline{\text{RTS}}$[1]. When in this mode, the receiver automatically negates $\overline{\text{RTS}}$ when a valid start bit is detected and the FIFO is full. When a FIFO position becomes available, the receiver asserts $\overline{\text{RTS}}$. Using this mode of operation prevents overrun errors by connecting the $\overline{\text{RTS}}$ to the $\overline{\text{CTS}}$ input of the transmitting device.

To use the $\overline{\text{RTS}}$ signals on UART1, the Pin Configuration Register in the SIM must be set up to enable the corresponding I/O pins for these functions. If the FIFO contains characters and the receiver is disabled, the CPU can still read the characters in the FIFO. If the receiver is reset, the FIFO and all receiver status bits, corresponding output ports, and interrupt request are reset. No additional characters are received until the receiver is re-enabled.

15.3.3 Looping Modes

The UART can be configured to operate in various looping modes as shown in Figure 15-7. These modes are useful for local and remote system diagnostic functions. The modes are described in the following paragraphs with additional information available in Section 15.4, “UART Memory Map and Register Definitions.”

Switching between modes should only be done while the transmitter and receiver are disabled because the selected mode is activated immediately on mode selection, even if this occurs in the middle of character transmission or reception. In addition, if a mode is deselected, the device switches out of the mode immediately, except for automatic echo and remote echo loopback modes. In these modes, the deselection occurs just after the receiver has sampled the stop bit (this is also the one-half point). For automatic echo mode, the transmitter stays in this mode until the entire stop bit has been retransmitted.

15.3.3.1 Automatic Echo Mode

In this mode, the UART automatically retransmits the received data on a bit-by-bit basis. The local CPU-to-receiver communication continues normally but the CPU-to-transmitter link is disabled. While in this mode, received data is clocked on the receiver clock and retransmitted on TxD. The receiver must be enabled but not the transmitter. Instead, the transmitter is clocked by the receiver clock.

Because the transmitter is not active, the TxEMP and TxRDY bits in USR are inactive and data is transmitted as it is received. Received parity is checked but not recalculated for transmission. Character framing is also checked but stop bits are transmitted as received. A received break is echoed as received until the next valid start bit is detected.

1. CTS and RTS are not available on UART2.

15.3.3.2 Local Loopback Mode

In this mode, TxD is internally connected to RxD. This mode is useful for testing the operation of a local UART module channel by sending data to the transmitter and checking data assembled by the receiver. In this manner, correct channel operations can be assured. Both transmitter and CPU-to-receiver communications continue normally in this mode. While in this mode, the RxD input data is ignored, the TxD is held marking, and the receiver is clocked by the transmitter clock. The transmitter must be enabled but not the receiver.

15.3.3.3 Remote Loopback Mode

In this mode, the channel automatically transmits received data on the TxD output on a bit-by-bit basis. The local CPU-to-transmitter link is disabled. This mode is useful for testing remote channel receiver and transmitter operation. While in this mode, the receiver clocks the transmitter.

NOTE

Because the receiver is not active, the CPU cannot read received data. All status conditions are inactive. Received parity is not checked and is not recalculated for transmission. Stop bits are transmitted as received. A received break is echoed as received until the next valid start bit is detected.

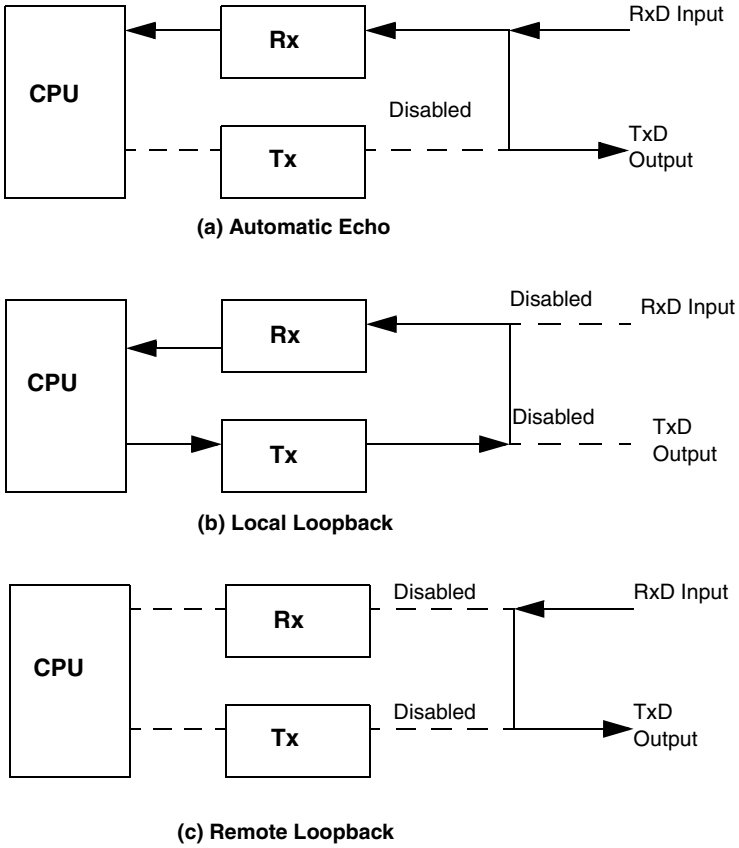


Figure 15-7. Looping Modes Functional Diagram

15.3.4 Multidrop Mode

The UART can be programmed to operate in a wakeup mode for multidrop or multiprocessor applications. Functional timing information for the multidrop mode is shown in Figure 15-8. The mode is selected by setting bits 3 and 4 in UART mode register 1 (UMR1). This mode of operation connects the master station to several slave stations (maximum of 256). In this mode, the master transmits an address character followed by a block of data characters targeted for one of the slave stations. The slave stations channel receivers are disabled; however, they continuously monitor the data stream sent out by the master station. When the master sends an address character, the slave receiver channel notifies its respective CPU by setting the RxRDY bit in the USR 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 wants to receive the subsequent data characters or block of data from the master station. Slave stations not addressed continue to monitor the data stream for the next address character. Data fields in the data stream are separated by an address character. After a slave receives a block of data, the slave station CPU disables the receiver and reinitiates the process.

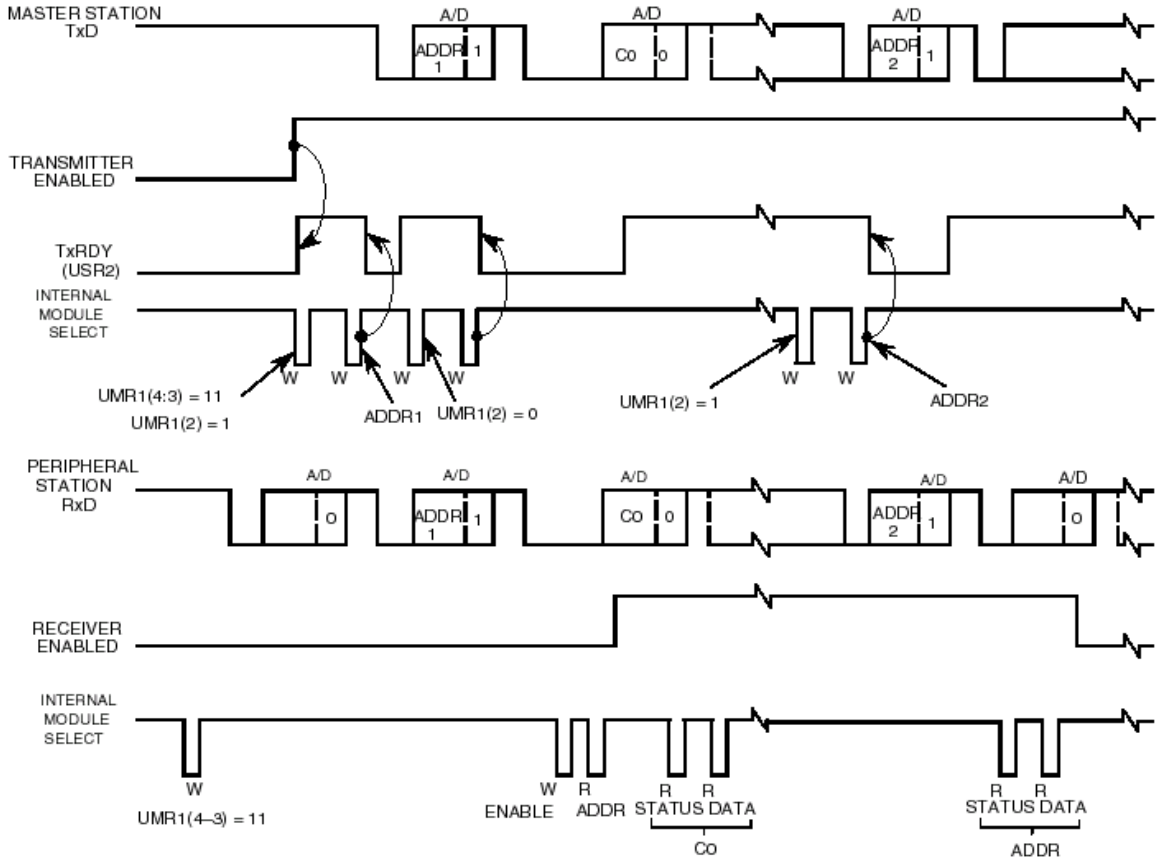


Figure 15-8. Multidrop Mode Timing Diagram

A transmitted character 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. The A/D bit identifies the type of character being transmitted to the slave station. The character is interpreted as an address character if the A/D bit is set or as a data character if the A/D bit is cleared. The polarity of the A/D bit is selected by

programming bit 2 of UMR1. UMR1 should also 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 one (address tag). The character is discarded if the received A/D bit is a zero (data tag). If the receiver is enabled, all received characters are transferred to the CPU using the receiver holding register stack during read operations.

In either case, the data bits are loaded into the data portion of the stack while the A/D bit is loaded into the status portion of the stack normally used for a parity error (USR bit 5). 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 can still contain error detection and correction information. One way to provide error detection, if 8-bit characters are not required, is to use software to calculate parity and append it to the 5-, 6-, or 7-bit character.

15.3.5 Bus Operation

This section describes the operation of the bus during read, write, and interrupt- acknowledge cycles to the UART module. All UART module registers must be accessed as bytes.

15.3.5.1 Read Cycles

The CPU accesses the UART module with 1 to 2 wait states because the core system clock is divided by 2 for the UART module. The UART module responds to reads with byte data on D[7:0]. Reserved registers return logic zero during reads.

15.3.5.2 Write Cycles

The CPU with zero wait states accesses the UART module. The UART module accepts write data on D[7:0]. Write cycles to read-only registers and reserved registers complete in a normal manner without exception processing; however, the data is ignored.

15.3.5.3 Interrupt Acknowledge Cycles

The UART module can arbitrate for interrupt servicing and supply the interrupt vector when it has successfully won arbitration. The vector number must be provided if interrupt servicing is necessary; thus, the interrupt vector register (UIVR) must be initialized. The interrupt vector number generated by the IVR is used if the autovector is not enabled in the SIM Interrupt Control Register (ICR). If the UIVR is not initialized and the ICR is not programmed for autovector, a spurious interrupt exception is taken if interrupts are generated. This works in conjunction with the MCF5251 interrupt controller, which allows a programmable Interrupt Priority Level (IPL) for the interrupt.

15.4 UART Memory Map and Register Definitions

This section contains a detailed description of each register and its specific function as well as flowcharts of basic UART module programming.

Writing control bytes into the appropriate registers controls the UART operation. A list of UART module registers and their associated addresses is shown in Table 15-1.

NOTE

All UART module registers are accessible only as bytes. The contents of the mode registers (UMR1 and UMR2), clock-select register (UCSR), and the auxiliary control register (UACR) bit 7 should be changed only after the receiver/transmitter is issued a software RESET command—i.e., channel operation must be disabled. Be careful if the register contents are changed during receiver/transmitter operations because unpredictable results can occur.

For the registers described in this section, the numbers above the register description represent the bit position in the register. The register description contains the mnemonic for the bit. The values as shown in the following tables are the values of those register bits after a hardware reset. A value of U indicates that the bit value is unaffected by reset. The read/write status is shown in the last line.

Table 15-1. UART Module Memory Map

UART 0	UART 1	UART2	Register Read (R/W = 1)	Register Write (R/W = 0)
MBAR+$1C0	MBAR+$200	MBAR2+$C00	Mode Register (UMR1, UMR2)	
MBAR+$1C4	MBAR+$204	MBAR2+$C04	Status Register (USR)	Clock-Select Register (UCSR)
MBAR+$1C8	MBAR+$208	MBAR2+$C08	DO NOT ACCESS[1]	Command Register (UCR)
MBAR+$1CC	MBAR+$20C	MBAR2+$C0C	Receiver Buffer (URB)	Transmitter Buffer (UTB)
MBAR+$1D0	MBAR+$210	MBAR2+$C10	Input Port Change Register (UIPCR)	Auxiliary Control Register (UACR)
MBAR+$1D4	MBAR+$214	MBAR2+$C14	Interrupt Status Register (UISR)	Interrupt Mask Register (UIMR)
MBAR+$1D8	MBAR+$218	MBAR2+$C18	Baud Rate Generator Prescale MSB (UBG1)	
MBAR+$1DC	MBAR+$21C	MBAR2+$C1C	Baud Rate Generator Prescale LSB (UBG2)	
DO NOT ACCESS[1]				
MBAR+$1F0	MBAR+$230	MBAR2+$C30	Interrupt Vector Register (UIVR)	
MBAR+$1F4	MBAR+$234	MBAR2+$C34	Input Port Register (UIP)	DO NOT ACCESS[1]
MBAR+$1F8	MBAR+$238	MBAR2+$C38	DO NOT ACCESS[1]	Output Port Bit Set CMD (UOP1)[2]
MBAR+$1FC	MBAR+$23C	MBAR2+$C3C	DO NOT ACCESS[1]	Output Port Bit Reset CMD (UOP0)[2]

1 This address is used for factory testing and should not be read. Reading this location results in undesired effects and possible incorrect transmission or reception of characters. Register contents can also be changed.

2 Address-triggered commands.

15.4.1 Mode Register 1 (UMR1*n*)

The UMR1 controls some of the UART module configuration. This register can be read or written at any time and is accessed when the mode register pointer points to UMR1. The pointer is set to UMR1 by RESET or by a set pointer command using the control register. After reading or writing UMR1, the pointer points to UMR2.

Address MBAR + $1C0 (UMR10)
MBAR + $200 (UMR11)
MBAR2 + $C00 (UMR12)

Access: Supervisor or User read/write

	7	6	5	4	3	2	1	0
R W	RXRTS	RXIRQ	ERR	PM1	PM0	PT	B/C1	B/C0
Reset	0	0	0	0	0	0	0	0

Figure 15-9. Mode Register 1 (UMR1*n*)

Table 15-2. Mode Register 1 (UMR1*n*) Field Descriptions

Field	Description
7 RxRTS	Receiver Request-to-Send Control 1 On receipt of a valid start bit, $\overline{RTS}$ is negated if the UART FIFO is full. $\overline{RTS}$ is reasserted when the FIFO has an empty position available. 0 The receiver has no effect on $\overline{RTS}$. The RTS is asserted by writing a one to the Output Port Bit Set Register (UOP1) This feature can be used for flow control to prevent overrun in the receiver by using the $\overline{RTS}$ output to control the $\overline{CTS}$ input of the transmitting device. If both the receiver and transmitter are programmed for $\overline{RTS}$ control, $\overline{RTS}$ control is disabled for both because such a configuration is incorrect. **Note:** Not available on UART2.
6 RxIRQ	RxIRQ—Receiver Interrupt Select 1 FFULL is the source that generates IRQ 0 RxRDY is the source that generates IRQ
5 ERR	The Error Mode bit controls the meaning of the three FIFO status bits (RB, FE, and PE) in the USR. 1 Block mode—The values in the channel USR are the accumulation (i.e., the logical OR) of the status for all characters coming to the top of the FIFO since the last reset error status command for the channel was issued. Refer to Section 15.4.17.1, “UART Module Initialization,” for more information on UART module commands. 0 Character mode—The values in the channel USR reflect the status of the character at the top of the FIFO. ERR = 0 must be used to obtain the correct A/$\overline{D}$ flag information when in multidrop mode.
4–3 PM	The Parity Mode bits encode the type of parity used for the channel (see Table 15-3). The parity bit is added to the transmitted character and the receiver performs a parity check on incoming data. These bits can alternatively select multidrop mode for the channel.

Table 15-2. Mode Register 1 (UMR1*n*) Field Descriptions (continued)

Field	Description
2 PT	The Parity Type bit selects the parity type if parity is programmed by the parity mode bits; if multidrop mode is selected, it configures the transmitter for data character transmission or address character transmission. Table 15-3 lists the parity mode and type or the multidrop mode for each combination of the parity mode and the parity type bits. **Table 15-3. PMx and PT Control Bits** PM1 / PM0 / Parity Mode / PT / Parity Types 0 / 0 / With Parity / 0 / Even Parity 0 / 0 / With Parity / 1 / Odd Parity 0 / 1 / Force Parity / 0 / Low Parity 0 / 1 / Force Parity / 1 / High Parity 1 / 0 / No Parity / X / No Parity 1 / 1 / Multidrop Mode / 0 / Data Character 1 / 1 / Multidrop Mode / 1 / Address Character “Force parity low” means forcing a 0 parity bit. “Force parity high” forces a 1 parity bit.
1–0 B/C	The Bits per Character bits select the number of data bits per character to be transmitted. The character length listed in Table 15-4 does not include start, parity, or stop bits. **Table 15-4. B/Cx Control Bits** B/C1 / B/C0 / Bits/Character 0 / 0 / 5 Bits 0 / 1 / 6 Bits 1 / 0 / 7 Bits 1 / 1 / 8 Bits

15.4.2 Mode Register 2 (UMR2*n*)

The UART mode register 2 (UMR2*n*) controls the 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*. Accesses to UMR2*n* do not update the pointer.

Address MBAR + $1C0 (UMR20)
MBAR + $200 (UMR21)
MBAR2 + $C00 (UMR22)

Access: Supervisor or User read/write

	7	6	5	4	3	2	1	0
R W	CM1	CM0	TxRTS	TxCTS	SB3	SB2	SB1	SB0
Reset	0	0	0	0	0	0	0	0

Table 15-5. Mode Register 2 (UMR2)

Table 15-6. Mode Register 2 (UMR2*n*) Field Descriptions

Field	Description
7–6 CM	Channel mode. Selects a channel mode. Section 15.3.3, "Looping Modes," describes individual modes. 00 Normal 01 Automatic echo 10 Local loop-back 11 Remote loop-back
5 TxRTS	Transmitter ready-to-send. Controls negation of RTS to automatically terminate a message transmission when the transmitter is disabled after completion of a transmission. Attempting to program a receiver and transmitter in the same channel for RTS control is not permitted and disables RTS control for both. 0 The transmitter has no effect on RTS. 1 When the transmitter is disabled after transmission completes, setting this bit automatically clears UOP[RTS] one bit time after any characters in the channel transmitter shift and holding registers are completely sent, including the programmed number of stop bits. **Note:** Not available on UART2.
4 TxCTS	Transmitter clear-to-send. If both TxCTS and TxRTS are enabled, TxCTS controls the operation of the transmitter. 0 CTS has no effect on the transmitter. 1 Enables clear-to-send operation. The transmitter checks the state of CTS each time it is ready to send a character. If CTS is asserted, the character is sent; if it is negated, the channel TxD remains in the high state and transmission is delayed until CTS is asserted. Changes in CTS as a character is being sent do not affect its transmission. **Note:** Not available on UART2
3–0 SB	Stop-bit length control. Selects the length of the stop bit appended to the transmitted character. Stop-bit lengths of 9/16th to 2 bits are programmable for 6–8 bit characters. Lengths of 1 1/16th 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, that is, 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 2 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 Bits	6–8 Bits
1000	1.563	1.563
1001	1.625	1.625
1010	1.688	1.688
1011	1.750	1.750
1100	1.813	1.813
1101	1.875	1.875
1110	1.938	1.938
1111	2.000	2.000

15.4.3 Status Registers (USR*n*)

The USR registers indicate the status of the characters in the receive FIFO and the status of the transmitter and receiver. The RB, FE, and PE bits are cleared by the Reset Error Status command in the UCR registers if the RB bit has not been read. Also, RB, FE, PE and OE can also be cleared by reading the Receive buffer (URB).

Address MBAR + $1C4 (USR0)
MBAR + $204 (USR1)
MBAR2 + $C04 (USR2)

Access: Supervisor or User read/write

	7	6	5	4	3	2	1	0
R W	RB	FE	PE	OE	TXEMP	TXRDY	FFULL	RXRDY
Reset	0	0	0	0	0	0	0	0

Figure 15-10. Status Registers (USR0 and USR1)

Table 15-7. Status Registers (USR*n*) Field Descriptions

Field	Description
7 RB	Received Break 1 An all-zero character of the programmed length has been received without a stop bit. The RB bit is valid only when the RxRDY bit is set. A single FIFO position is occupied when a break is received. Additional entries into the FIFO are inhibited until RxD returns to the high state for at least one-half bit time, which is equal to two successive edges of the internal or external clock x 1 or 16 successive edges of the external clock x 16. The received break circuit detects breaks that originate in the middle of a received character. However, if a break begins in the middle of a character, it must persist until the end of the next detected character time. 0 No break has been received.
6 FE	Framing Error 1 A stop bit was not 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. The bit is valid only when the RxRDY bit is set. 0 No framing error has occurred.
5 PE	Parity Error 1 When the with-parity or force-parity mode is programmed in the UMR1, the corresponding character in the FIFO was received with incorrect parity. When the multidrop mode is programmed, this bit stores the received A/D bit. This bit is valid only when the RxRDY bit is set. 0 No parity error has occurred.
4 OE	Overrun Error 1 One or more characters in the received data stream have been lost. This bit is set on 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 the UCR clears this bit. 0 No overrun has occurred.
3 TxEMP	Transmitter Empty 1 The transmitter has underrun (both 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. 0 The transmitter buffer is not empty. Either a character is currently being shifted out or the transmitter is disabled. Users can enable/disable the transmitter by programming the TCx bits in the UCR.
2 TxRDY	Transmitter Ready 1 The transmitter-holding register is empty and ready to be loaded with a character. This bit is set when the character is transferred to the transmitter shift register. This bit is also set when the transmitter is first enabled. Characters loaded into the transmitter holding register while the transmitter is disabled are not transmitted. 0 The CPU has loaded the transmitter-holding register or the transmitter is disabled.

Table 15-7. Status Registers (USR*n*) Field Descriptions (continued)

Field	Description
1 FFULL	FIFO Full 1 Three characters have been received and are waiting in the receiver buffer FIFO. 0 The FIFO is not full but can contain as many as two unread characters.
0 RxRDY	Receiver Ready 1 One or more characters have been received and are waiting in the receiver buffer FIFO. 0 The CPU has read the receiver buffer and no characters remain in the FIFO after this read.

15.4.4 Clock-Select Registers (USCR*n*)

To use the timer mode for either the receiver and transmitter channel, program the UCSR registers to the value $DD.

NOTE

External clock input is not available so this register should always be set to select internal timer mode.

The transmitter and receiver can be programmed to different clock sources.

Address MBAR + $1C4 (USCR0
MBAR + $204 (USCR1)
MBAR2 + $C04 (USCR2)

Access: User write only

	7	6	5	4	3	2	1	0
R								
W	RCS3	RCS2	RCS1	RCS0	TCS3	TCS2	TCS1	TCS0
Reset	1	1	0	1	1	1	0	1

Figure 15-11. Clock Select Register (UCSR*n*)

Table 15-8. Clock Select Register (UCSR*n*) Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7–4
RCS3–RCS0</td><td>The Receiver Clock Select bits select the clock source for the receiver channel. Table 15-12 details the register bits necessary for each mode.
<table>
<tr><th>RCS3</th><th>RCS2</th><th>RCS1</th><th>RCS0</th><th>Mode</th></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>TIMER</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>reserved</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>reserved</td></tr>
</table></td></tr>
<tr><td>3–0
TCS3–TCS0</td><td>The Transmitter Clock Select bits determine the clock source of the UART transmitter channel.
<table>
<tr><th>TCS3</th><th>TCS2</th><th>TCS1</th><th>TCS0</th><th>SET 1</th></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>TIMER</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>reserved</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>reserved</td></tr>
</table></td></tr>
</table>

15.4.5 Command Registers (UCR*n*)

The UCR supplies commands to the UART. Multiple commands can be specified in a single write to the UCR if the commands are not conflicting. For example, reset-transmitter and enable-transmitter commands cannot be specified in a single command.

Address MBAR + $1C8 (UCR0)
MBAR + $208 (UCR1)
MBAR2 + $C08 (UCR2)

Access: User write only

	7	6	5	4	3	2	1	0
R								
W		MISC2	MISC1	MISC0	TC1	TC0	RC1	RC0
Reset	0	0	0	0	0	0	0	0

Figure 15-12. Command Register (UCR*n*)

15.4.5.1 Miscellaneous Commands

Bits MISC3 through MISC0 select a single command as listed in Table 15-9.

Table 15-9. MISC*x* Control Bits

MISC2	MISC1	MISC0	Command
0	0	0	No Command
0	0	1	Reset Mode Register Pointer
0	1	0	Reset Receiver
0	1	1	Reset Transmitter
1	0	0	Reset Error Status
1	0	1	Reset Break-Change Interrupt
1	1	0	Start Break
1	1	1	Stop Break

15.4.5.1.1 Reset Mode Register Pointer

The reset mode register pointer command causes the mode register pointer to point to UMR1.

15.4.5.1.2 Reset Receiver

The reset receiver command resets the receiver. The receiver is immediately disabled, the FFULL and RxRDY bits in the USR are cleared, and the receiver FIFO pointer is reinitialized. All other registers are unaltered. Use this command instead of the receiver-disable command whenever the receiver configuration is changed (it places the receiver in a known state).

15.4.5.1.3 Reset Transmitter

The reset transmitter command resets the transmitter. The transmitter is immediately disabled and the TxEMP and TxRDY bits in the USR are cleared. All other registers are unaltered. Use this command instead of the transmitter-disable command whenever the transmitter configuration is changed (it places the transmitter in a known state).

15.4.5.1.4 Reset Error Status

The reset error status command clears the RB, FE, PE, and OE bits in the USR. This command is also used in the block mode to clear all error bits after a data block is received.

15.4.5.1.5 Reset Break-Change Interrupt

The reset break-change interrupt command clears the delta break (DBx) bit in the UISR.

15.4.5.1.6 Start Break

The start break command forces TxD low. If the transmitter is empty, the start of the break conditions can be delayed by as much as two bit times. If the transmitter is active, the break begins when transmission of the character is complete. If a character is in the transmitter shift register, the start of the break is delayed until the character is transmitted. If the transmitter holding register has a character, that character is transmitted before the break. The transmitter must be enabled for this command to be accepted. The state of the $\overline{\text{CTS}}$ input is ignored for this command.

15.4.5.1.7 Stop Break

The stop break command causes TxD to go high (mark) within two bit times. Characters stored in the transmitter buffer, if any, are transmitted.

15.4.5.2 Transmitter Commands

Bits TC1 and TC0 select a single command as listed in Table 15-10.

Table 15-10. TC*x* Control Bits

TC1	TC0	Command
0	0	No Action Taken
0	1	Transmitter Enable
1	0	Transmitter Disable
1	1	Do Not Use

15.4.5.2.1 No Action Taken

The “no action taken” command causes the transmitter to stay in its current mode. If the transmitter is enabled, it remains enabled; if disabled, it remains disabled.

15.4.5.2.2 Transmitter Enable

The “transmitter enable” command enables operation of the channel's transmitter. The TxEMP and TxRDY bits in the USR are also set. If the transmitter is already enabled, this command has no effect.

15.4.5.2.3 Transmitter Disable

The “transmitter disable” command terminates transmitter operation and clears the TxEMP and TxRDY bits in the USR. However, if a character is being transmitted when the transmitter is disabled, the transmission of the character is completed before the transmitter becomes inactive. If the transmitter is already disabled, this command has no effect.

15.4.5.2.4 Do Not Use

Do not use this bit combination because the result is indeterminate.

15.4.5.3 Receiver Commands

Bits RC1 and RC0 select a single command as listed in Table 15-11.

Table 15-11. RC*x* Control Bits

RC1	RC0	Command
0	0	No Action Taken
0	1	Receiver Enable
1	0	Receiver Disable
1	1	Do Not Use

15.4.5.3.1 No Action Taken

The “no action taken” command causes the receiver to stay in its current mode. If the receiver is enabled, it remains enabled; if disabled, it remains disabled.

15.4.5.3.2 Receiver Enable

The “receiver enable” command enables operation of the channel's receiver. If the UART module is not in multidrop mode, this command also forces the receiver into the search-for-start-bit state. If the receiver is already enabled, this command has no effect.

15.4.5.3.3 Receiver Disable

The “receiver disable” command immediately disables the receiver. Any character being received is lost. The command has no effect on the receiver status bits or any other control register. If the UART module is programmed to operate in the local loopback mode or multidrop mode, the receiver operates even though this command is selected. If the receiver is already disabled, this command has no effect.

15.4.5.3.4 Do Not Use

Do not use this bit combination because the result is indeterminate.

15.4.6 Receiver Buffer Registers (UBR*n*)

The receiver buffer (URB) contains three receiver-holding registers and a serial shift register. The RxD pin is connected to the serial shift register while the holding registers act as a FIFO. The CPU reads from the top of the stack while the receiver shifts and updates from the bottom of the stack when the shift register has been filled (see Figure 15-4).

Address MBAR + $1CC (URB0)
MBAR + $20C (URB1)
MBAR2 + $C0C (URB2)

Access: User read only

	7	6	5	4	3	2	1	0
R	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
W								
Reset	1	1	1	1	1	1	1	1

Figure 15-13. Receiver Buffer (URB*n*) Register

Table 15-12. Receiver Buffer (URB*n*) Register Field Descriptions

Field	Description
7–0 RB7–RB0	These bits contain the character in the receiver buffer.

15.4.7 Transmitter Buffer Registers (UTB*n*)

The transmitter buffer (UTB) consists of two registers: the transmitter-holding register and the transmitter shift register (see Figure 15-4). The holding register accepts characters from the bus master if the TxRDY bit in the channel's USR is set. A write to the transmitter buffer clears the TxRDY bit, inhibiting additional characters until the shift register is ready to accept more data. When the shift register is empty, it checks the holding register for a valid character to be sent (TxRDY bit cleared). If a valid character is present, the shift register loads the character and reasserts the TxRDY bit in the USR. Writes to the transmitter buffer when the channel's UART Status Register (USR) TxRDY bit is clear and when the transmitter is disabled have no effect on the transmitter buffer.

Address MBAR + $1CC (UTB0)
MBAR + $20C (UTB1)
MBAR2 + $C0C (UTB2)

Access: User write only

	7	6	5	4	3	2	1	0
R								
W	TB7	TB6	TB5	TB4	TB3	TB2	TB1	TB0
Reset	0	0	0	0	0	0	0	0

Figure 15-14. Transmitter Buffer (UTB*n*) Register

Table 15-13. Transmitter Buffer (UTB*n*) Register Field Descriptions

Bit Name	Description
TB7–TB0	These bits contain the character in the transmitter buffer.

15.4.8 Input Port Change Registers (UIPCR*n*)

The UIPCR registers show the current state and the change-of-state for the $\overline{\text{CTS}}$ pin. (Note: not available on UART2).

Address MBAR + $1D0 (UIPCR0)
MBAR + $210 (UIPCR1)
MBAR2 + $C10 (UIPCR2)

Access: User read only

	7	6	5	4	3	2	1	0
R				COS				$\overline{\text{CTS}}$
W								
Reset	0	0	0	0	1	1	1	$\overline{1}$

Figure 15-15. Input Port Change Register (UIPCR*n*)

Table 15-14. Input Port Change Register (UIPCR*n*) Field Descriptions

Field	Description
7–5, 3–1	Reserved
4 COS	Change-of-State 1 A change-of-state (high-to-low or low-to-high transition), lasting longer than 25–50 μs has occurred at the $\overline{\text{CTS}}$ input. When this bit is set, the UART Auxiliary Control Register (UACR) can be programmed to generate an interrupt to the CPU. 0 No change-of-state has occurred since the last time the CPU read the UART Input Port Change Register (UIPCR). A read of the UIPCR also clears the UART Interrupt Status Register (UISR)COS bit.
0 $\overline{\text{CTS}}$	Current State Starting two serial clock periods after reset, the $\overline{\text{CTS}}$ bit reflects the state of the $\overline{\text{CTS}}$ pin. If the $\overline{\text{CTS}}$ pin is detected as asserted at that time, the COS bit is set, which initiates an interrupt if the Input Enable Control (IEC) bit of the UACR register is enabled. 1 The current state of the $\overline{\text{CTS}}$ input is logic one. 0 The current state of the $\overline{\text{CTS}}$ input is logic zero. **Note:** Not available on UART2

15.4.9 Auxiliary Control Registers (UACR*n*)

The UART auxiliary control registers control the input enable.

Address MBAR + $1D0 (UACR0)
MBAR + $210 (UACR1)
MBAR2 + $C10 (UACR2)

Access: User write only

	7	6	5	4	3	2	1	0
R								
W								IEC
Reset	0	0	0	0	0	0	0	0

Figure 15-16. Auxiliary Control Register (UACR*n*)

Table 15-15. Auxiliary Control Register (UACR*n*) Field Descriptions

Field	Description
7–1	Reserved
0 IEC	Input Enable Control 1 UISR bit 7 is set and generates an interrupt when the COS bit in the UART Input Port Change Register (UIPCR) is set by an external transition on the $\overline{\text{CTS}}$ input (if bit 7 of the interrupt mask register (UIMR) is set to enable interrupts). 0 Setting the corresponding bit in the UIPCR has no effect on UISR bit 7.

15.4.10 Interrupt Status Registers (UISR*n*)

The UISR registers provides status for all potential interrupt sources. The UART Interrupt Mask Register (UIMR) masks the contents of this register. If a flag in the UISR is set and the corresponding bit in UIMR is also set, the internal interrupt output is asserted. If the corresponding bit in the UIMR is cleared, the state of the bit in the UISR has no effect on the interrupt output.

NOTE

The UIMR does not mask reading of the UISR. True status is provided regardless of the contents of UIMR. A UART module reset clears the contents of UISR.

Address MBAR + $1D4 (UISR0)
MBAR + $214 (UISR1)
MBAR2 + $C14 (UISR2)

Access: User read only

	7	6	5	4	3	2	1	0
R	COS					DB	RXRDY	TXRDY
W								
Reset	0	0	0	0	0	0	0	0

Figure 15-17. Interrupt Status Register (UISR*n*)

Table 15-16. Interrupt Status Register (UISR*n*) Field Descriptions

Field	Description
7 COS	Change-of-State 1 A change-of-state has occurred at the $\overline{\text{CTS}}$ input and has been selected to cause an interrupt by programming bit 0 of the UACR. 0 COS bit in the UIPCR is not selected.
6–3	Reserved
2 DB	Delta Break 1 The receiver has detected the beginning or end of a received break. 0 No new break-change condition to report. Refer to Section 15.4.5, "Command Registers (UCRn)," for more information on the reset break-change interrupt command.
1 RxRDY	Receiver Ready or FIFO Full UMR1 bit 6 programs the function of this bit. It is a duplicate of either the FFULL or RxRDY bit of USR.
0 TxRDY	Transmitter Ready This bit is the duplication of the TxRDY bit in USR. 1 The transmitter holding register is empty and ready to be loaded with a character. 0 The CPU loads the transmitter-holding register or the transmitter is disabled. Characters loaded into the transmitter-holding register when TxRDY=0 are not transmitted.

15.4.11 Interrupt Mask Registers (UIMR*n*)

The UIMR registers select the corresponding bits in the UISR that cause an interrupt. By setting the bit, the interrupt is enabled. If one of the bits in the UISR is set and the corresponding bit in the UIMR is also set, the internal interrupt output is asserted. If the corresponding bit in the UIMR is zero, the state of the bit in the UISR has no effect on the interrupt output. The UIMR does not mask the reading of the UISR.

Address MBAR + $1D4 (UIMR0)
MBAR + $214 (UIMR1)
MBAR2 + $C14 (UIMR2)

Access: User write only

	7	6	5	4	3	2	1	0
R								
W	COS					DB	FFULL	TXRDY
Reset	0	0	0	0	0	0	0	0

Figure 15-18. Interrupt Mask Register (UIMR*n*)

Table 15-17. Interrupt Mask Register (UIMR*n*) Field Descriptions

Field	Description
7 COS	Change-of-State 1 Enable interrupt 0 Disable interrupt
6–3	Reserved
2 DB	Delta Break 1 Enable interrupt 0 Disable interrupt

Table 15-17. Interrupt Mask Register (UIMR*n*) Field Descriptions (continued)

Field	Description
1 FFULL	FIFO Full 1 Enable interrupt 0 Disable interrupt
0 TxRDY	Transmitter Ready 1 Enable interrupt 0 Disable interrupt

15.4.12 Baud Rate Generator (MSB) Register (UBG1*n*)

The UBG1n register hold the eight most significant bits of the preload value the timer uses for providing a given baud rate. The minimum value that can be loaded on the concatenation of UBG1 with UBG2 is $0002. This register is write only and cannot be read by the CPU.

15.4.13 Baud Rate Generator (LSB) Register (UBG2*n*)

The UBG2n register holds the eight least significant bits of the preload value the timer uses for providing a given baud rate. The minimum value that can be loaded on the concatenation of UBG1 with UBG2 is $0002. This register is write only and cannot be read by the CPU.

15.4.14 Interrupt Vector Registers (UIVR*n*)

The UIVR registers contain the 8-bit vector number of the internal interrupt.

Address MBAR + $1F0 (UIVR0)
MBAR + $230 (UIVR1)
MBAR2 + $C30 (UIVR2)

Access: Supervisor or User read/write

	7	6	5	4	3	2	1	0
R W	IVR7	IVR6	IVR5	IVR4	IVR3	IVR2	IVR1	IVR0
Reset	0	0	0	0	1	1	1	1

Figure 15-19. Interrupt Vector Register (UIVR*n*)

Table 15-18. Interrupt Vector Register (UIVR*n*) Field Descriptions

Field	Description
7–0 IVR7–IVR0	The Interrupt Vector Bits are an 8-bit number that indicates the offset from the base of the vector table where the address of the exception handler for the specified interrupt is located. The UIVR is reset to $0F, which indicates an uninitialized interrupt condition.

15.4.15 Input Port Registers (UIP*n*)

The UIP registers show the current state of the $\overline{\text{CTS}}$ input.

Address MBAR + $1F4 (UIP0)
MBAR + $234 (UIP1)
MBAR2 + $C34 (UIP2)

Access: Supervisor or User read only

	7	6	5	4	3	2	1	0
R								$\overline{\text{CTS}}$
W								
Reset	1	1	1	1	1	1	1	1

Figure 15-20. Input Port Register (UIP*n*)

Table 15-19. Interrupt Port Register (UIP*n*) Field Descriptions

Bit Name	Description
7–1	Reserved
0 $\overline{\text{CTS}}$	Current State 1 The current state of the $\overline{\text{CTS}}$ input is logic one. 0 The current state of the $\overline{\text{CTS}}$ input is logic zero. The information contained in this bit is latched and reflects the state of the input pin at the time that the UIP is read. This bit has the same function and value as the UIPCR bit 0.

15.4.16 Output Port Data Registers (UOP1*n*)

The RTS output is set by a bit set command (writing to UOP1n) and is cleared by a bit reset command (writing to UOP0n). (Note: not available on UART2).

Address MBAR + $1F8 (UOP10)
MBAR + $238 (UOP11)
MBAR2 + $C38 (UOP12)

Access: User write only

	7	6	5	4	3	2	1	0
R								
W								$\overline{\text{RTS}}$
Reset	–	–	–	–	–	–	–	0

Figure 15-21. Output Port Data Registers (UOP1*n*)

Table 15-20. Output Port Data Registers (UOP1*n*) Field Descriptions

Field	Description
7–1	Reserved
$\overline{\text{RTS}}$	Output Port Parallel Output 1 A write cycle to the OPset address asserts the $\overline{\text{RTS}}$ signal. 0 This bit is not affected by writing a zero to this address. The output port bits are inverted at the pins so the $\overline{\text{RTS}}$ set bit provides an asserted $\overline{\text{RTS}}$ pin. **Note:** Not available on UART2.

Address MBAR + $1FC (UOP00)
MBAR + $23C (UOP01)
MBAR2 + $C3C (UOP02)

Access: User write only

	7	6	5	4	3	2	1	0
R								
W								RTS
Reset	–	–	–	–	–	–	–	–

Figure 15-22. Output Port Data Registers (UOP0*n*)

15.4.17 Programming

Figure 15-23 shows the basic interface software flowchart required for operation of the UART module. The routines are divided into these three categories:

1. UART Module Initialization
2. I/O Driver
3. Interrupt Handling

15.4.17.1 UART Module Initialization

The UART module initialization routines consist of SINIT and CHCHK. SINIT is called at system initialization time to check UART operation. Before SINIT is called, the calling routine allocates two words on the system stack. On return to the calling routine, SINIT passes information on the system stack to reflect the status of the UART. If SINIT finds no errors, the receiver and transmitter are enabled. The CHCHK routine performs the actual checks as called from the SINIT routine. When called, SINIT places the UART in the local loopback mode and checks for the following errors:

- Transmitter Never Ready
- Receiver Never Ready
- Parity Error
- Incorrect Character Received

15.4.17.2 I/O Driver Example

The I/O driver routines consist of INCH and OUTCH. INCH is the terminal input character routine and obtains a character from the receiver. OUTCH sends a character to the transmitter.

15.4.17.3 Interrupt Handling

The interrupt-handling routine consists of SIRQ, 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.

15.5 UART Module Initialization Sequence

The following steps are required to properly initialize the UART module:

Command Register (UCR)

Reset the receiver and transmitter.

Baud Rate Generator Register (UBG1 and UBG2)

Set Baud Rate.

Interrupt Vector Register (UIVR)

Program the vector number for a UART module interrupt or use auto-vector, as desired.

Interrupt Mask Register (UIMR)

Enable the desired interrupt sources.

Auxiliary Control Register (UACR)

Initialize the Input Enable Control (IEC) bit.

Clock Select Register (UCSR)

Select the receiver and transmitter internal clock.

Mode Register 1 (UMR1)

1. If required, program operation of Receiver Ready-to-Send (RxRTS Bit).
2. Select Receiver-Ready or FIFO-Full Notification (R/F Bit).
3. Select character or block-error mode (ERR Bit).
4. Select parity mode and type (PM and PT Bits).
5. Select number of bits per character (B/Cx Bits).

Mode Register 2 (UMR2)

1. Select the mode of operation (CMx bits).
2. If required, program operation of Transmitter Ready-to-Send (TxRTS Bit).
3. If required, program operation of Clear-to-Send (TxCTS Bit).
4. Select stop-bit length (SBx Bits).

Command Register (UCR)

Enable the receiver and transmitter.

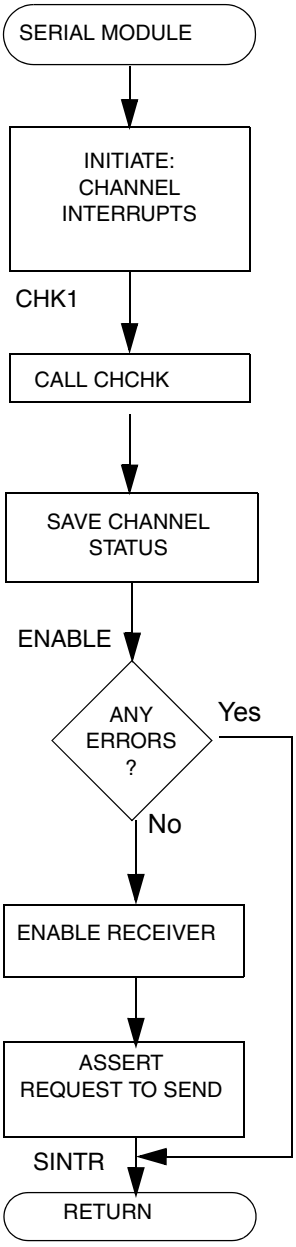


Figure 15-23. UART Software Flowchart (1 of 5)

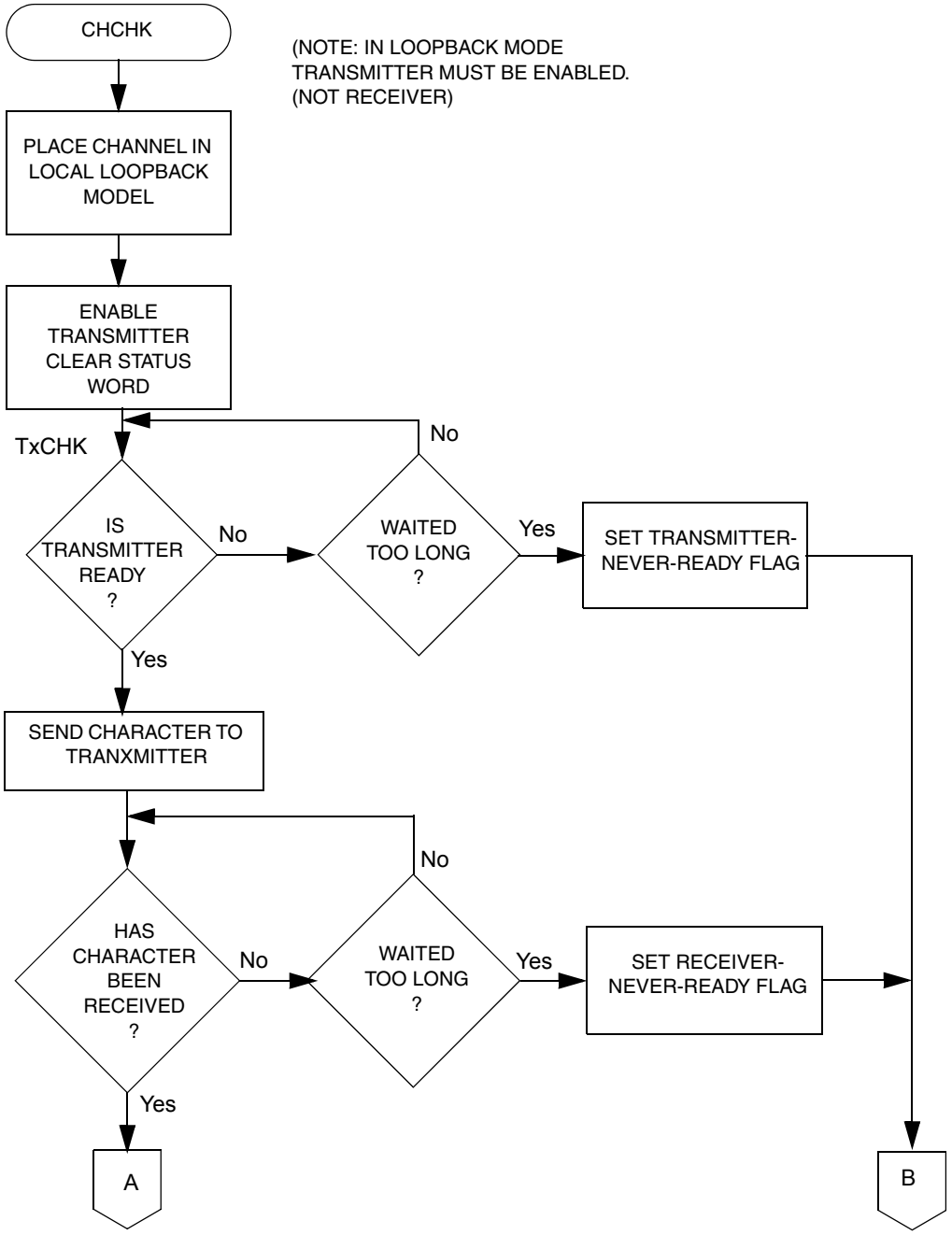


Figure 15-24. UART Software Flowchart (2 of 5)

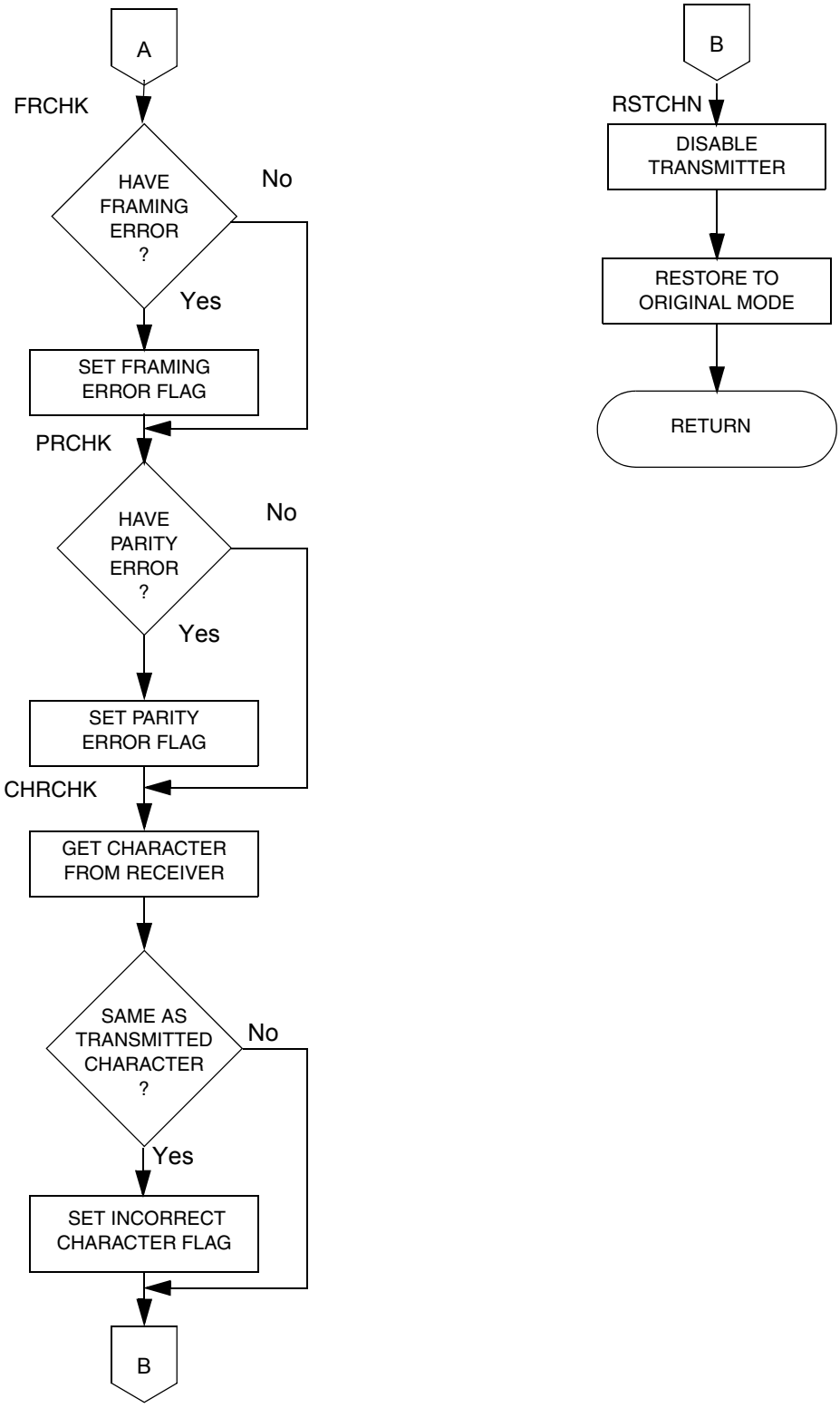


Figure 15-25. UART Software Flowchart (3 of 5)

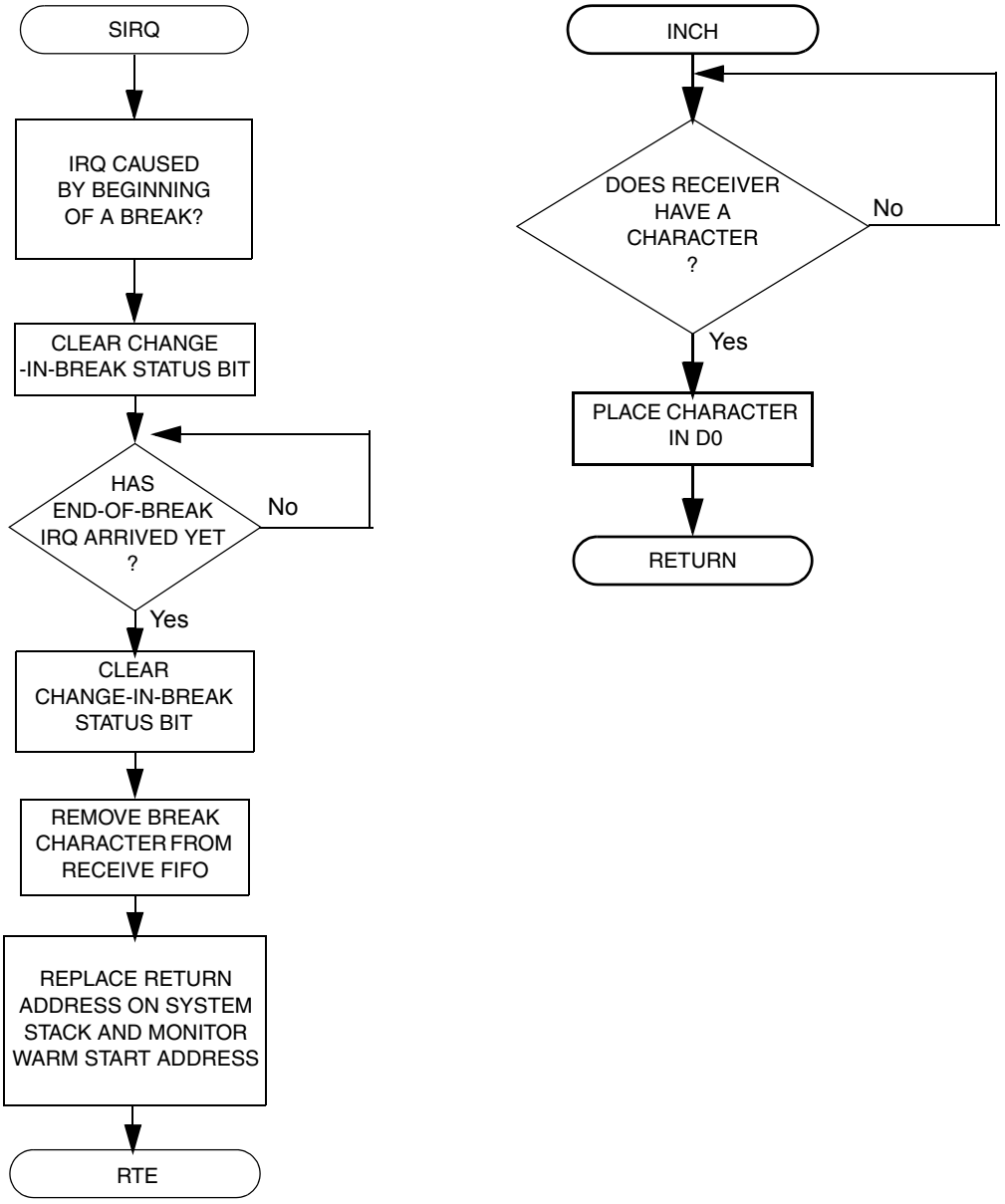


Figure 15-26. UART Software Flowchart (4 of 5)

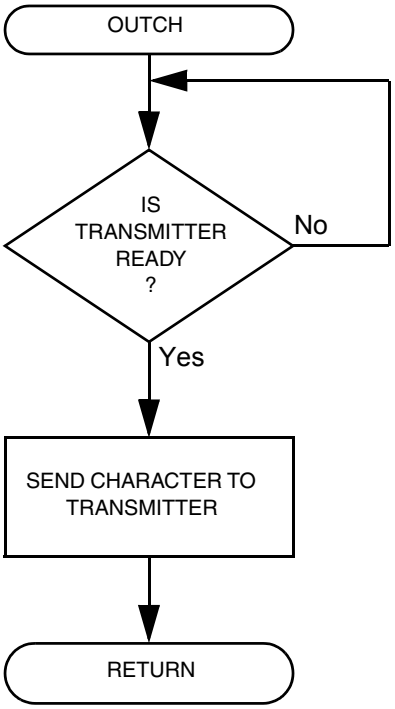


Figure 15-27. UART Software Flowchart (5 of 5)

Chapter 16
Queued Serial Peripheral Interface (QSPI) Module

This chapter describes the operation of the Queued Serial Peripheral interface module of the MCF5251 and provides its memory map and register descriptions.

16.1 Features

The QSPI module provides a serial peripheral interface with queued transfer capability. It allows users to queue up to 16 transfers at once, eliminating CPU intervention between transfers.

- Programmable queue to support up to 16 transfers without user intervention
- Supports transfer sizes of 8 to 16 bits in 1-bit increments
- Four peripheral chip-select lines for control of up to 15 devices
- Programmable baud rates up to 17.5Mbps at a CPU clock of 140 MHz
- Programmable delays
- Programmable clock phase and polarity
- Supports wraparound mode for continuous transfers

16.2 QSPI Module Overview

The QSPI module communicates with the core using internal memory mapped registers starting at MBAR + $400. See Section 16.4, "QSPI Memory Map and Register Definitions." A block diagram of the QSPI module is shown in Figure 16-1.

16.2.1 Interface and Pins

The module supports 4 external CS pins which can be decoded externally to provide control for up to 15 devices. There are a total of seven signals: QSPI_Dout, QSPI_Din, QSPI_CLK, QSPI_CS [3:0].

Peripheral chip-select signals, QSPI_CS[3:0], are used to select an external device as the source or destination for serial data transfer. Signals are asserted at a logic level corresponding to the value of the QSPI_CS[3:0] bits in the command RAM whenever a command in the queue is executed. More than one chip-select signal can be asserted simultaneously.

Although QSPI_CS[3:0] will function as simple chip selects in most applications, up to 15 devices can be selected by decoding them with an external 4-to-16 decoder.

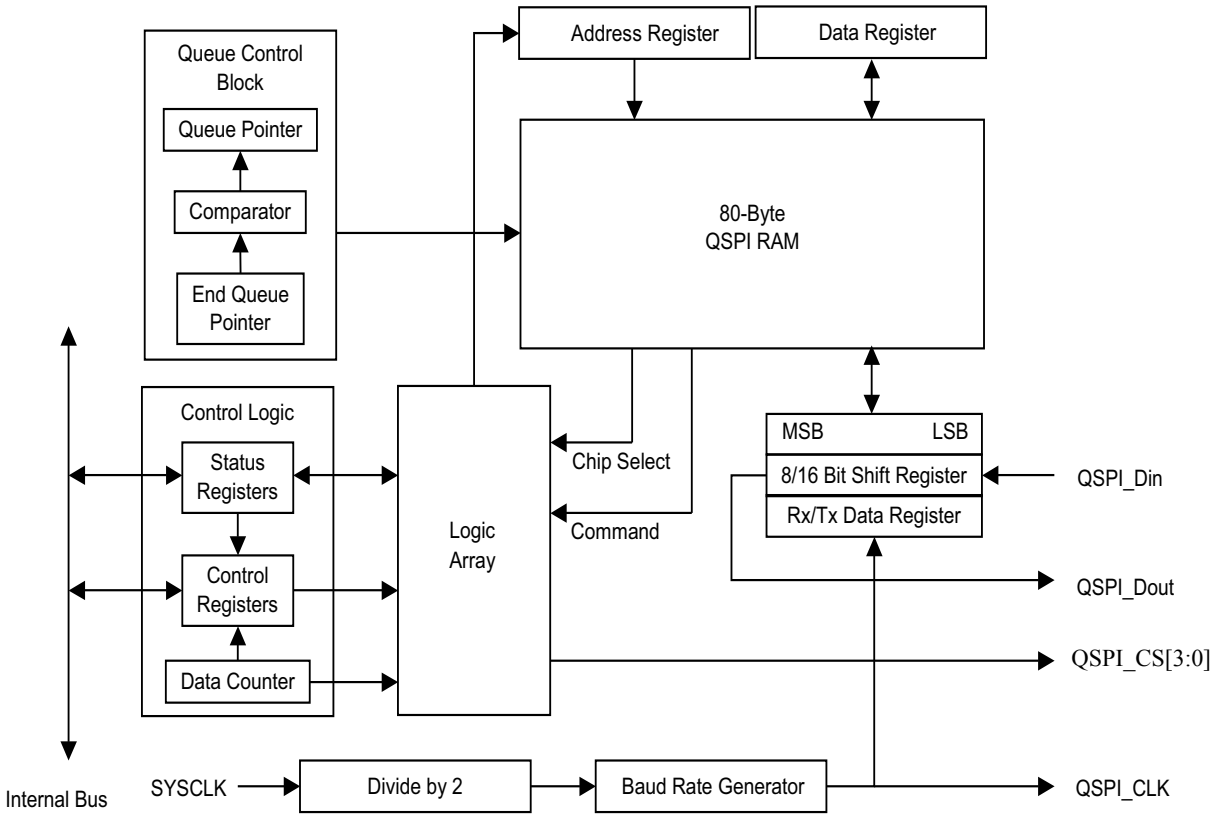


Figure 16-1. QSPI Block Diagram

Table 16-1. QSPI Input and Output Signals and Functions

Signal Name	Hi_Z or Actively Driven	Function
QSPI Data Output (QSPI_Dout)	Configurable	Serial data output from QSPI
QSPI Data Input (QSPI_Din)	N/A	Serial data input to QSPI
Serial Clock (QSPI_CLK)	Actively driven	Clock output from QSPI
Peripheral Chip Selects (QSPI_CS[3:0])	Actively driven	Peripheral selects

16.2.2 Internal Bus Interface

Because the QSPI module only operates in master mode, the master bit in the QSPI mode register (QMR[MSTR]) must be set for the QSPI to function properly. The QSPI can initiate serial transfers but cannot respond to transfers initiated by other QSPI masters.

16.3 Operation

The QSPI uses a dedicated 80-byte block of static RAM accessible both to the module and the CPU to perform queued operations. The RAM is divided into three segments as follows:

- 16 command control bytes (command RAM)

- 16 transmit data words (transfer RAM)
- 16 receive data words (transfer RAM)

RAM is organized so that 1 byte of command control data, 1 word of transmit data, and 1 word of receive data comprise 1 queue entry.

The user initiates QSPI operation by loading a queue of commands in command RAM, writing transmit data into transmit RAM, and then enabling the QSPI data transfer. The QSPI executes the queued commands and sets the completion flag in the QSPI interrupt register (QIR[SPIF]) to signal their completion. Optionally, QIR[SPIFE] can be enabled to generate an interrupt.

The QSPI uses four queue pointers. The user can access three of them through fields in QSPI Wrap Register (QWR):

- The new queue pointer, QWR[NEWQP], points to the first command in the queue.
- An internal queue pointer points to the command currently being executed.
- The completed queue pointer, QWR[CPTQP], points to the last command executed.
- The end queue pointer, QWR[ENDQP], points to the final command in the queue.

The internal pointer is initialized to the same value as QWR[NEWQP]. During normal operation, the following sequence repeats:

1. The command pointed to by the internal pointer is executed.
2. The value in the internal pointer is copied into QWR[CPTQP].
3. The internal pointer is incremented.

Execution continues at the internal pointer address unless the QWR[NEWQP] value is changed. After each command is executed, QWR[ENDQP] and QWR[CPTQP] are compared. When a match occurs, QIR[SPIF] is set and the QSPI stops unless wraparound mode is enabled. Setting QWR[WREN] enables wraparound mode.

QWR[NEWQP] is cleared at reset. When the QSPI is enabled, execution begins at address 0x0 unless another value has been written into QWR[NEWQP]. QWR[ENDQP] is cleared at reset but is changed to show the last queue entry before the QSPI is enabled. QWR[NEWQP] and QWR[ENDQP] can be written at any time. When the QWR[NEWQP] value changes, the internal pointer value also changes unless a transfer is in progress, in which case the transfer completes normally. Leaving QWR[NEWQP] and QWR[ENDQP] set to 0x0 causes a single transfer to occur when the QSPI is enabled.

Data is transferred relative to QSPI_CLK which can be generated in any one of four combinations of phase and polarity using QMR[CPHA, CPOL]. Data is transferred most significant bit (msb) first. The number of bits transferred defaults to eight, but can be set to any value from 8 to 16 by writing a value into the BITSE field of the command RAM, QCR[BITSE].

16.3.1 QSPI RAM

The QSPI contains an 80-byte block of static RAM that can be accessed by both the user and the QSPI. This RAM does not appear in the MCF5251 memory map because it can only be accessed by the user indirectly through the QSPI address register (QAR) and the QSPI data register (QDR).

The RAM is divided into three segments with 16 addresses each:

- Receive data RAM, the initial destination for all incoming data
- Transmit data RAM, a buffer for all out-bound data
- Command RAM, where commands are loaded

The transmit and command RAM are write-only by the user. The receive RAM is read-only by the user. Figure 16-2 shows the RAM configuration. The RAM contents are undefined immediately after a reset.

The command and data RAM in the QSPI is indirectly accessible with QDR and QAR as 48 separate locations that comprise 16 words of transmit data, 16 words of receive data and 16 bytes of commands.

A write to QDR causes data to be written to the RAM entry specified by QAR[ADDR] and causes the value in QAR to increment. Correspondingly, a read at QDR returns the data in the RAM at the address specified by QAR[ADDR]. This also causes QAR to increment. A read access requires a single wait state.

Relative Address	Register	Function
0x00	QTR0	Transmit RAM
0x01	QTR1	
.	.	16 bits wide
.	.	
.	.	
0x0F	QTR15	
0x10	QRR0	Receive RAM
0x11	QRR1	
.	.	16 bits wide
.	.	
.	.	
0x1F	QRR15	
0x20	QCR0	Command RAM
0x21	QCR1	
.	.	8 bits wide
.	.	
.	.	
0x2F	QCR15	

Figure 16-2. QSPI RAM Model

16.3.1.1 Transmit RAM

Data to be transmitted by the QSPI is stored in the transmit RAM segment located at addresses 0x0 to 0xF. The user normally writes 1 word into this segment for each queue command to be executed. The user cannot read transmit RAM.

Out-bound data must be written to transmit RAM in a right-justified format. The unused bits are ignored. The QSPI copies the data to its data serializer (shift register) for transmission. The data is transmitted most significant bit first and remains in transmit RAM until overwritten by the user.

16.3.1.2 Receive RAM

Data received by the QSPI is stored in the receive RAM segment located at 0x10 to 0x1F in the QSPI RAM space. The user reads this segment to retrieve data from the QSPI. Data words with less than 16 bits are stored right-justified in the RAM. Unused bits in a receive queue entry are set to zero upon completion of the individual queue entry.

NOTE

Throughout ColdFire documentation, ‘word’ is used consistently and exclusively to designate a 16-bit data unit. The only exceptions to this rule appear in the sections that detail serial communication modules such as the QSPI that supports variable-length data units. To simplify this issue, the functional unit is referred to as a ‘word’ regardless of length.

QWR[CPTQP] shows which queue entries have been executed. The user can query this field to determine which locations in receive RAM contain valid data.

16.3.1.3 Command RAM

The CPU writes one byte of control information to this segment for each QSPI command to be executed. Command RAM is write-only memory from a user’s perspective.

Command RAM consists of 16 bytes with each byte divided into two fields. The peripheral chip select field controls the QSPI_CS signal levels for the transfer. The command control field provides transfer options.

A maximum of 16 commands can be in the queue. Queue execution proceeds from the address in QWR[NEWQP] through the address in QWR[ENDQP].

The QSPI executes a queue of commands defined by the control bits in each command RAM entry which sequence the following actions:

- chip-select pins are activated
- data is transmitted from transmit RAM and received into the receive RAM
- the synchronous transfer clock QSPI_CLK is generated

Before any data transfers begin, control data must be written to the command RAM, and any out-bound data must be written to transmit RAM. Also, the queue pointers must be initialized to the first and last entries in the command queue.

Data transfer is synchronized with the internally generated QSPI_CLK, whose phase and polarity are controlled by QMR[CPHA] and QMR[CPOL]. These control bits determine which QSPI_CLK edge is used to drive outgoing data and to latch incoming data.

16.3.2 Baud Rate Selection

Baud rate is selected by writing a value from 2 to 255 into QMR[BAUD]. The QSPI uses a prescaler to derive the QSPI_CLK rate from the system clock, SYSCLK, divided by two.

A baud rate value of zero turns off the QSPI_CLK.

The desired QSPI_CLK baud rate is related to SYSCLK and QMR[BAUD] by the following expression:

QMR[BAUD] = SYSCLK / [2 × (desired QSPI_CLK baud rate)] (SYSCLK = CORE operating frequency / 2).

Table 16-2. QSPI_CLK Frequency as Function of SYSCLK and Baud Rate

QMR [BAUD]	SYSCLK			
	70 MHz	48 MHz	33 MHz	20 MHz
2	17,500,000	12,000,000	8,250,000	5,000,000
4	8,750,000	6,000,000	4,125,000	2,500,000
8	4,375,000	3,000,000	2,062,500	1,250,000
16	2,187,500	1,500,000	1,031,250	625,000
32	1,093,750	750,000	515,625	312,500
255	546,875	94,118	64,706	39,216

16.3.3 Transfer Delays

The QSPI supports programmable delays for the QSPI_CS signals. The time between QSPI_CS assertion and the leading QSPI_CLK edge, and the time between the end of one transfer and the beginning of the next, are both independently programmable.

The chip select to clock delay enable (DSCK) bit in command RAM, QCR[DSCK], enables the programmable delay period from QSPI_CS assertion until the leading edge of QSPI_CLK. QDLYR[QCD] determines the period of delay before the leading edge of QSPI_CLK. The following expression determines the actual delay before the QSPI_CLK leading edge:

QSPI_CS-to-QSPI_CLK delay = QCD/SYSCLK frequency

QCD has a range of 1 to 127

When QCD or DSCK equals zero, the standard delay of one-half the QSPI_CLK period is used.

The delay after transmit enable (DT) bit in command RAM enables the programmable delay period from the negation of the QSPI_CS signals until the start of the next transfer. The delay after transfer can be used to provide a peripheral deselect interval. A delay can also be inserted between consecutive transfers to allow serial A/D converters to complete conversion. There are two transfer delay options: the user can

choose to delay a standard period after serial transfer is complete or can specify a delay period. Writing a value to QDLYR[DTL] specifies a delay period. The DT bit in command RAM determines whether the standard delay period (DT = 0) or the specified delay period (DT = 1) is used. The following expression is used to calculate the delay:

Delay after transfer = 32 × QDLYR[DTL] /SYSCLK frequency (DT = 1)

where QDLYR[DTL] has a range of 2 to 255.

A zero value for DTL causes a delay-after-transfer value of 8192/SYSCLK frequency.

Standard delay after transfer = 17/SYSCLK frequency (DT = 0)

Adequate delay between transfers must be specified for long data streams because the QSPI module requires time to load a transmit RAM entry for transfer. Receiving devices need at least the standard delay between successive transfers. If SYSCLK is operating at a slower rate, the delay between transfers must be increased proportionately.

16.3.4 Transfer Length

There are two transfer length options. The user can choose a default value of 8 bits or a programmed value of 8 to 16 bits inclusive. The programmed value must be written into QMR[BITS]. The bits per transfer enable (BITSE) field in the command RAM determines whether the default value (BITSE = 0) or the BITS[3–0] value (BITSE = 1) is used. QMR[BITS] gives the required number of bits to be transferred.

16.3.5 Data Transfer

Operation is initiated by setting QDLYR[SPE]. Shortly after QDLYR[SPE] is set, the QSPI executes the command at the command RAM address pointed to by QWR[NEWQP]. Data in transmit RAM is loaded into the data shift register and transmitted. Data that is simultaneously received is stored in the receive RAM.

When the proper number of bits has been transferred, the QSPI stores the working queue pointer value in QWR[CPTQP], increments the working queue pointer, and loads the next data for transfer from the transmit RAM. The command pointed to by the incremented working queue pointer is executed next unless a new value has been written to QWR[NEWQP]. If a new queue pointer value is written while a transfer is in progress, then that transfer is completed normally.

When the CONT bit in the command RAM is set, the QSPI_CS signals are asserted between transfers. When CONT is cleared, QSPI_CS[3:0] are negated between transfers. The QSPI_CS signals are not high impedance.

When the QSPI reaches the end of the queue, it asserts QIR[SPIF]. If QIR[SPIFE] is set, an interrupt request is generated when QIR[SPIF] is asserted. Then the QSPI clears QDLYR[SPE] and stops, unless wraparound mode is enabled.

Wraparound mode is enabled by setting QWR[WREN]. The queue can wrap to pointer address 0x0, or to the address specified by QWR[NEWQP], depending on the state of QWR[WRTO].

In wraparound mode, the QSPI cycles through the queue continuously, even while requesting interrupt service. QDLYR[SPE] is not cleared when the last command in the queue is executed. New receive data overwrites previously received data in the receive RAM. Each time the end of the queue is reached, QIR[SPIFE] is set. QIR[SPIF] is not automatically reset. If interrupt driven QSPI service is used, the service routine must clear QIR[SPIF] to abort the current request. Additional interrupt requests during servicing can be prevented by clearing QIR[SPIFE].

There are two recommended methods of exiting wraparound mode: clearing QWR[WREN] or setting QWR[HALT]. Exiting wraparound mode by clearing QDLYR[SPE] is not recommended because this may abort a serial transfer in progress. The QSPI sets SPIF, clears QDLYR[SPE], and stops the first time it reaches the end of the queue after QWR[WREN] is cleared. After QWR[HALT] is set, the QSPI finishes the current transfer, then stops executing commands. After the QSPI stops, QDLYR[SPE] can be cleared.

16.4 QSPI Memory Map and Register Definitions

The programming model for the QSPI consists of six registers. They are the QSPI mode register (QMR), QSPI delay register (QDLYR), QSPI wrap register (QWR), QSPI interrupt register (QIR), QSPI address register (QAR), and the QSPI data register (QDR).

There are a total of 80 bytes of memory used for transmit, receive, and control data. This memory is accessed indirectly using QAR and QDR.

Registers and RAM are written and read by the CPU.

16.4.1 QSPI Mode Register (QMR)

The QMR register, shown in Figure 16-3, determines the basic operating modes of the QSPI module. Parameters such as clock polarity and phase, baud rate, master mode operation, and transfer size are determined by this register. The data output high impedance enable, DOHIE, controls the operation of QSPI_Dout between data transfers. When DOHIE is cleared, QSPI_Dout is actively driven between transfers. When DOHIE is set, QSPI_Dout assumes a high impedance state.

NOTE

Because the QSPI does not operate in slave mode, the master mode enable bit, QMR[MSTR], must be set for the QSPI module to operate correctly.

Address MBAR + 0x400 Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	MSTR	DOHIE	BITS				CPOL	CPHA	BAUD							
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0

Figure 16-3. QSPI Mode Register (QMR)

Table 16-3. QSPI Mode Register (QMR) Field Descriptions

Field	Description
15 MSTR	Master mode enable. 0 Reserved, do not use. 1 The QSPI is in master mode. Must be set for the QSPI module to operate correctly.
14 DOHIE	Data output high impedance enable. Selects QSPI_Dout mode of operation. 0 Default value after reset. QSPI_Dout is actively driven between transfers. 1 QSPI_Dout is high impedance between transfers.
13–10 BITS	Transfer size—Determines the number of bits to be transferred for each entry in the queue. *Value ..... Bits per transfer* 0000 ..... 16 0001–0111 .... Reserved 1000 ..... 8 1001 ..... 9 1010 ..... 10 1011 ..... 11 1100 ..... 12 1101 ..... 13 1110 ..... 14 1111 15
9 CPOL	Clock polarity. Defines the clock polarity of QSPI_CLK. 0 The inactive state value of QSPI_CLK is logic level 0. 1 The inactive state value of QSPI_CLK is logic level 1.
8 CPHA	Clock phase. Defines the QSPI_CLK clock-phase. 0 Data is captured on the rising edge of QSPI_CLK and changed on the falling edge of QSPI_CLK. 1 Data is changed on the falling leading edge of QSPI_CLK and captured on the rising edge of QSPI_CLK.
7–0 BAUD	Baud rate divider. The baud rate is selected by writing a value in the range 2–255. A value of zero disables the QSPI. The desired QSPI_CLK baud rate is related to SYSCLK and QMR[BAUD] by the following expression: • QMR[BAUD] = SystemClock / [2 × (desired QSPI_CLK baud rate)]

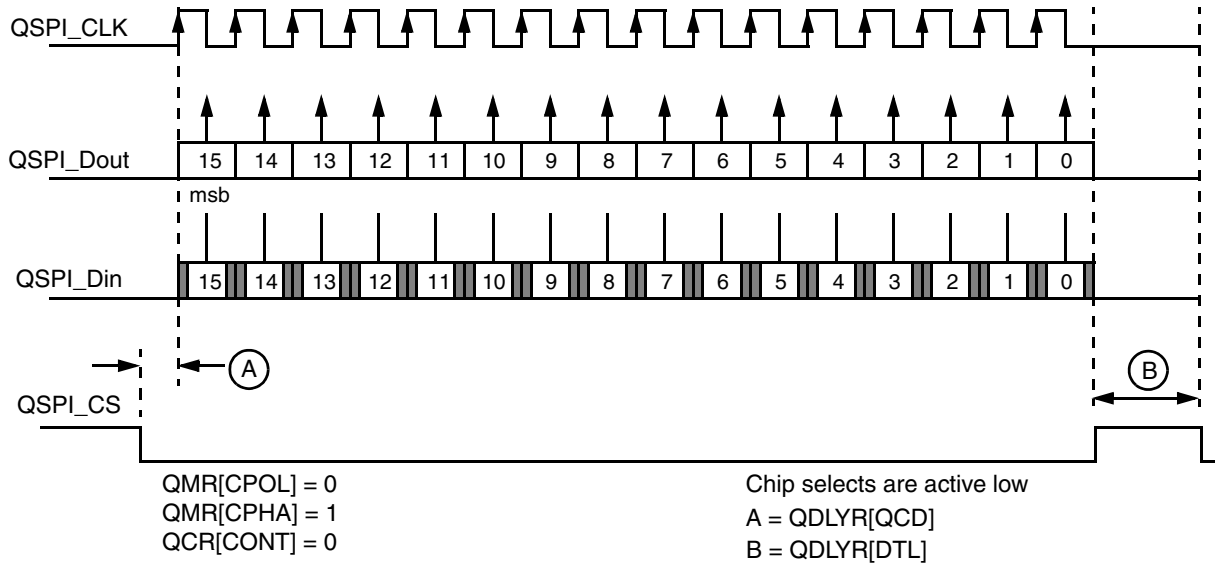


Figure 16-4. QSPI Clocking and Data Transfer Example

16.4.2 QSPI Delay Register (QDLYR)

Address MBAR + 0x404 Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	SPE	QCD							DTL							
Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0

Figure 16-5. QSPI Delay Register (QDLYR

Table 16-4. QSPI Delay Register (QDLYR) Field Descriptions

Field	Description
15 SPE	QSPI enable. When set, the QSPI initiates transfers in master mode by executing commands in the command RAM. Automatically cleared by the QSPI when a transfer completes.The user can also clear this bit to abort transfer unless QIR[ABRTL] is set. The recommended method for aborting transfers is to set QWR[HALT].
14–8 QCD	QSPILCK Delay. When the DSCK bit in the command RAM, is set this field determines the length of the delay from assertion of the chip selects to valid QSPI_CLK transition.
7–0 DTL	Delay after transfer. When the DT bit in the command RAM sets this field, it determines the length of delay after the serial transfer.

16.4.3 QSPI Wrap Register (QWR)

Address MBAR + 0x408 Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	HALT	WREN	WRTO	CSIV	ENDQP				CPTQP				NEWQP			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-6. QSPI Wrap Register (QWR)

Table 16-5. QSPI Wrap Register (QWR) Field Descriptions

Field	Description
15 HALT	Halt transfers. Assertion of this bit causes the QSPI to stop execution of commands once it has completed execution of the current command.
14 WREN	Wraparound enable. Enables wraparound mode. 0 Execution stops after executing the command pointed to by QWR[ENDQP]. 1 After executing command pointed to by QWR[ENDQP], wrap back to entry zero, or the entry pointed to by QWR[NEWQP] and continue execution.
13 WRTO	Wraparound location. Determines where the QSPI wraps to in wraparound mode. 0 Wrap to RAM entry zero. 1 Wrap to RAM entry pointed to by QWR[NEWQP].

Table 16-5. QSPI Wrap Register (QWR) Field Descriptions (continued)

Field	Description
12 CSIV	QSPI_CS inactive level. 0 QSPI chip select outputs return to zero when not driven from the value in the current command RAM entry during a transfer (that is, inactive state is 0, chip selects are active high). 1 QSPI chip select outputs return to one when not driven from the value in the current command RAM entry during a transfer (that is, inactive state is 1, chip selects are active low).
11–8 ENDQP	End of queue pointer. Points to the RAM entry that contains the last transfer description in the queue.
7–4 CPTQP	Completed queue entry pointer. Points to the RAM entry that contains the last command to have been completed. This field is read only.
3–0 NEWQP	Start of queue pointer. This 4-bit field points to the first entry in the RAM to be executed on initiating a transfer.

16.4.4 QSPI Interrupt Register (QIR)

Address MBAR + 0x40C — Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WCEFB	ABRTB		ABRTL	WCEFE	ABRTE		SPIFE					WCEF	ABRT		SPIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-7. QSPI Interrupt Register (QIR)

Table 16-6. QSPI Interrupt Register (QIR) Field Descriptions

Field	Description
15 WCEFB	Write collision access error enable. A write collision occurs during a data transfer when the RAM entry containing the command currently being executed is written to by the CPU with the QDR. When this bit is asserted, the write access to QDR results in an access error.
14 ABRTB	Abort access error enable. An abort occurs when QDLYR[SPE] is cleared during a transfer. When set, an attempt to clear QDLYR[SPE] during a transfer results in an access error.
13	Reserved, should be cleared.
12 ABRTL	Abort lock-out. When set, QDLYR[SPE] cannot be cleared by writing to the QDLYR. QDLYR[SPE] is only cleared by the QSPI when a transfer completes.
11 WCEFE	Write collision interrupt enable. Interrupt enable for WCEF. Setting this bit enables the interrupt, and clearing it disables the interrupt.
10 ABRTE	Abort interrupt enable. Interrupt enable for ABRT flag. Setting this bit enables the interrupt, and clearing it disables the interrupt.
9	Reserved, should be cleared.
8 SPIFE	QSPI finished interrupt enable. Interrupt enable for SPIF. Setting this bit enables the interrupt, and clearing it disables the interrupt.
7–4	Reserved, should be cleared.

Table 16-6. QSPI Interrupt Register (QIR) Field Descriptions (continued)

Field	Description
3 WCEF	Write collision error flag. Indicates that an attempt has been made to write to the RAM entry that is currently being executed. Writing a 1 to this bit clears it and writing 0 has no effect.
2 ABRT	Abort flag. Indicates that QDLYR[SPE] has been cleared by the user writing to the QDLYR rather than by completion of the command queue by the QSPI. Writing a 1 to this bit clears it and writing 0 has no effect.
1	Reserved, should be cleared.
0 SPIF	QSPI finished flag. Asserted when the QSPI has completed all the commands in the queue. Set on completion of the command pointed to by QWR[ENDQP], and on completion of the current command after assertion of QWR[HALT]. In wraparound mode, this bit is set every time the command pointed to by QWR[ENDQP] is completed. Writing a 1 to this bit clears it and writing 0 has no effect.

The command and data RAM in the QSPI is indirectly accessible with QDR and QAR as 48 separate locations that comprise 16 words of transmit data, 16 words of receive data and 16 bytes of commands.

A write to QDR causes data to be written to the RAM entry specified by QAR[ADDR]. This also causes the value in QAR to increment.

Correspondingly, a read at QDR returns the data in the RAM at the address specified by QAR[ADDR]. This also causes QAR to increment. A read access requires a single wait state.

NOTE

The QAR does not wrap after the last queue entry within each section of the RAM.

16.4.5 QSPI Address Register (QAR)

The QAR, shown in Figure 16-8, is used to specify the location in the QSPI RAM that read and write operations affect.

Address MBAR + 0x410 Access: User read/write

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

Figure 16-8. QSPI Address Register (QAR)

16.4.6 QSPI Data Register (QDR)

The QDR, shown in Figure 16-9, is used to access QSPI RAM indirectly. The CPU reads and writes all data from and to the QSPI RAM through this register.

Address MBAR + 0x414 Access: User read/write

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

Figure 16-9. QSPI Data Register (QDR)

16.4.7 Command RAM Registers (QCR0–QCR15)

The command RAM is accessed using the upper byte of QDR. The QSPI cannot modify information in command RAM.

There are 16 bytes in the command RAM. Each byte is divided into two fields. The chip select field enables external peripherals for transfer. The command field provides transfer operations.

NOTE

The command RAM is accessed only using the most significant byte of QDR and indirect addressing based on QAR[ADDR].

Address QAR[ADDR] Access: User write only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	CONT	BITSE	DT	DSCK	QSPI_CS											
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 16-10. Command RAM Registers (QCR0–QCR15)

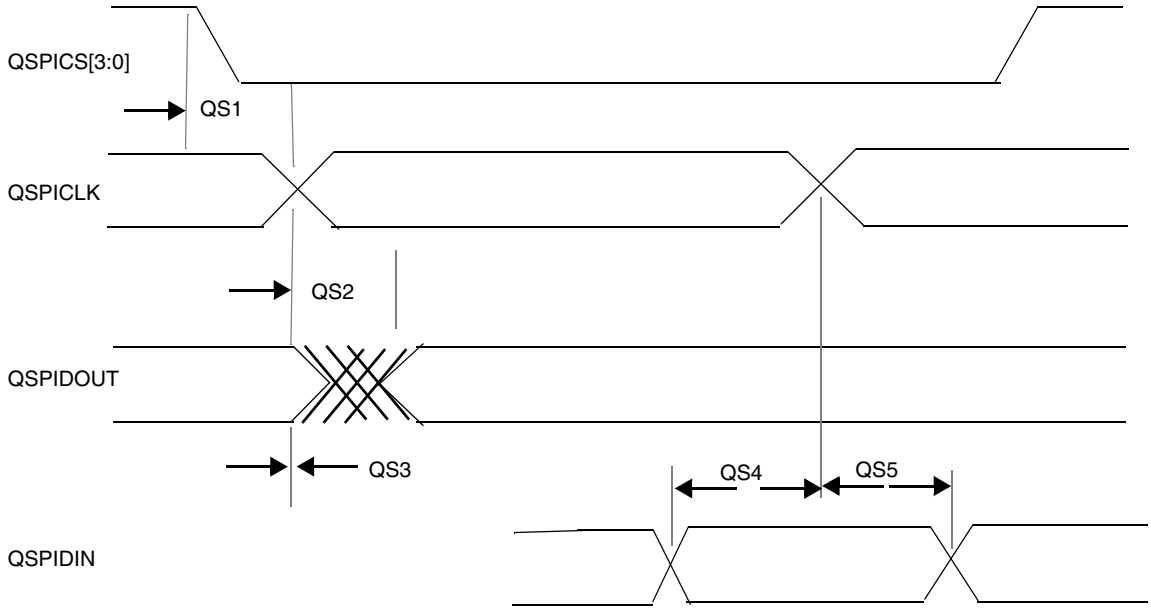
Table 16-7. Command RAM Registers (QCR*n*) Field Descriptions

Field	Description
15 CONT	Continuous. 0 Chip selects return to inactive level defined by QWR[CSIV] when transfer is complete. 1 Chip selects remain asserted after the transfer of 16 words of data[11].
14 BITSE	Bits per transfer enable. 0 Eight bits 1 Number of bits set in QMR[BITS]
13 DT	Delay after transfer enable. 0 Default reset value. 1 The QSPI provides a variable delay at the end of serial transfer to facilitate interfacing with peripherals that have a latency requirement. The delay between transfers is determined by QDLYR[DTL].
12 DSCK	Chip select to QSPI_CLK delay enable. 0 Chip select valid to QSPI_CLK transition is one-half QSPI_CLK period. 1 QDLYR[QCD] specifies the delay from QSPI_CS valid to QSPI_CLK.

Table 16-7. Command RAM Registers (QCR*n*) Field Descriptions (continued)

Field	Description
11–8 QSPI_CS	Peripheral chip selects. Used to select an external device for serial data transfer. More than one chip select may be active at once, and more than one device can be connected to each chip select.
7–0	Reserved, should be cleared.

1 To keep the chip selects asserted for all transfers, the QWR [CSIV] bit must be set to control the level that the chip selects return to after the first transfer.



	Min	Max
QS1: QSPICS to QSPICLK	1T1	
QS2: QSPI_CLK to QSPIDOUT VALID		20 ns
QS3: QSPI_CLK to QSPIDOUT HOLD	0 ns	
QS4: QSPI_DIN to QSPICLK SETUP	10 ns	
QS5: QSPI_DIN to QSPICLK HOLD	10 ns	

1 T1 is defined as the clock period in ns.

Figure 16-11. QSPI Timing

16.4.8 Programming Example

The following steps are necessary to set up the QSPI 12-bit data transfers and a QSPI_CLK of 15MHz. The QSPI RAM is set up for a queue of 16 transfers. All four QSPI_CS signals are used in this example.

1. Set QSPI pin functionality by the programming the PIN_CONFIG register as appropriate.

2. Write the QMR with 0xB302 to set up 12-bit data words with the data shifted on the falling clock edge, and a clock frequency of 15MHz (assuming a 60-MHz SYSCLK).
3. Write QDLYR with the desired delays.
4. Write QIR with 0xD00F to enable write collision, abort bus errors, and clear any interrupts.
5. Write QAR with 0x0020 to select the first command RAM entry.
6. Write QDR with 0x7E00, 0x7E00, 0x7E00, 0x7E00, 0x7D00, 0x7D00, 0x7D00, 0x7D00, 0x7B00, 0x7B00, 0x7B00, 0x7B00, 0x7700, 0x7700, 0x7700, and 0x7700 to set up four transfers for each chip select. The chip selects are active low in this example.
7. Write QAR with 0x0000 to select the first transmit RAM entry.
8. Write QDR with sixteen 12-bit words of data.
9. Write QWR with 0x0F00 to set up a queue beginning at entry 0 and ending at entry 15.
10. Set QDLYR[SPE] to enable the transfers.
11. Wait until the transfers are complete. QIR[SPIF] is set when the transfers are complete.
12. Write QAR with 0x0010 to select the first receive RAM entry.
13. Read QDR to get the received data for each transfer.
14. Repeat steps 5 through 13 to do another transfer.

Chapter 17
Audio Interface Module (AIM)

This chapter discusses the audio interface structure, memory map, and register descriptions, as well as transmit and receive interfaces.

17.1 Audio Interface Overview

The audio interface module provides the necessary input and output features to receive and transmit digital audio signals over serial audio interfaces (IIS/EIAJ) and over digital audio interfaces (IEC958).

The MCF5251 is equipped with three serial audio interfaces compliant with Philips I^2S and Sony EIAJ format. There are two IEC958 (SPDIF) receivers with 4 multiplexed inputs, and one IEC958 (SPDIF) transmitter with two outputs: One for professional C-channel and one for consumer C-channel.

The audio interface module allows the direct retransmission of an audio signal received on one receiver to another transmitter, without CPU intervention, or it allows the CPU to receive or transmit digital audio to or from any of the audio interfaces.

The IEC958 (SPDIF) receivers and transmitter support audio and allow the handling of IEC958 C and U channels.

A frequency measurement block exists to allow precise measurement of an incoming sampling frequency. This can be used in conjunction with the XTRIM output (and with the appropriate control software) to “lock” the clock that is being input to CRIN (either external generated clock or crystal) to the recovered SPDIF audio clock, if so desired. Some external hardware is required for this including a set of varicap diodes. This is covered in Section 17.9, “Phase/Frequency Determination and XTRIM Function.”

17.1.1 Audio Interface Block Diagram

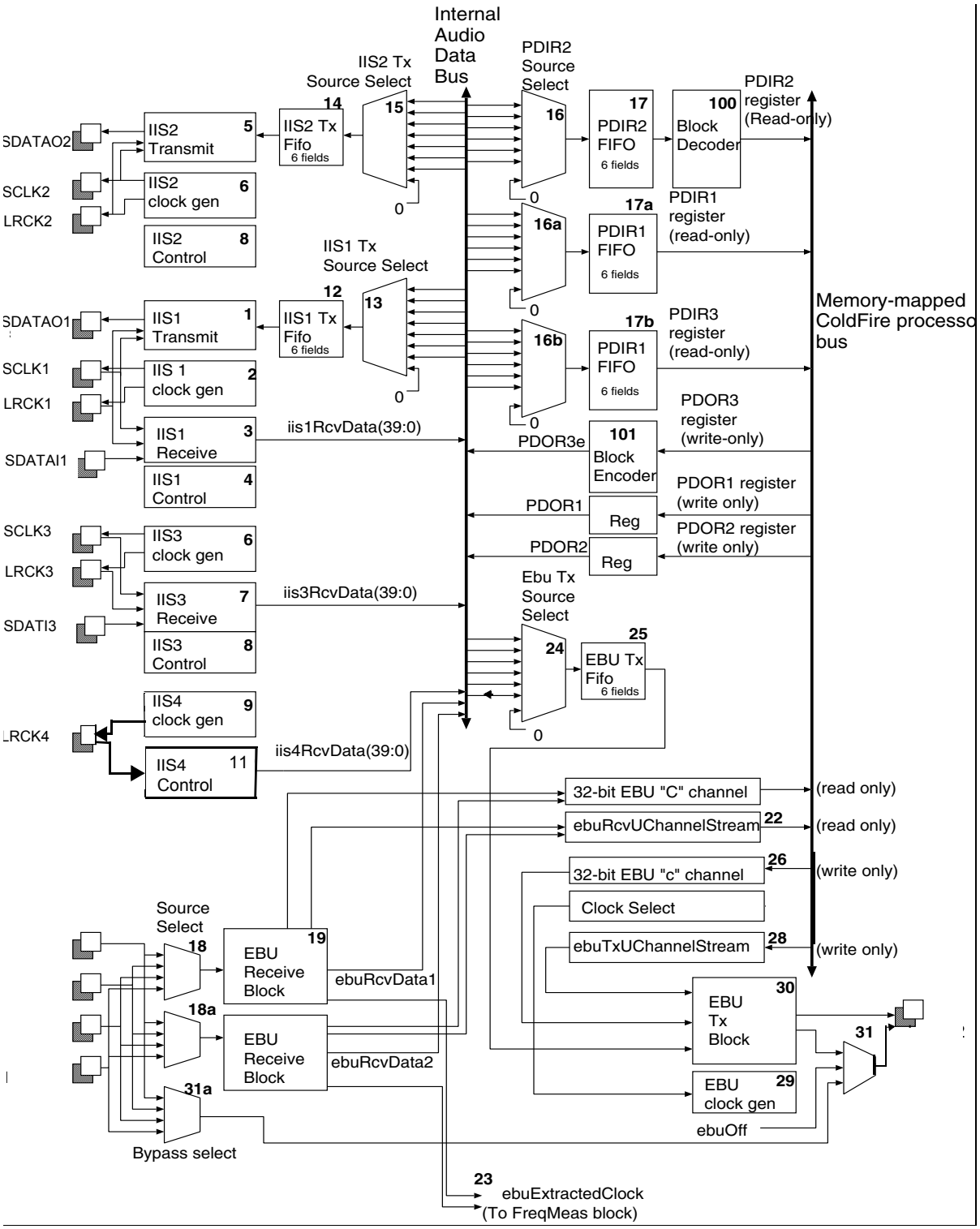


Figure 17-1. Audio Interface Block Diagram

17.1.2 Audio Interface Structure

The discussion in this section accompanies Figure 17-1 by identifying the blocks called out (in bold) in the diagram.

There are four serial audio interface blocks (**5–11**) labeled as follows:

1. IIS1: Capable of transmitting and receiving audio data.
2. IIS2: Transmit only.
3. IIS3: Receive only.
4. IIS4: Only SCLK4 input / output is available. This is intended to be used with the ADC circuit under certain conditions see Chapter 12, “Analog to Digital Converter (ADC)” for details.

As shown in Figure 17-1, there two IEC958 receivers. The source selector (**18)** and the receiver block itself (**19**). The receiver is capable of taking its input signal from four possible EBU inputs:

1. EBUIN1
2. EBUIN2
3. EBUIN3
4. EBUIN4

There is one IEC958 transmitter with two outputs (**30**), one carries the “consumer” C-channel, the other the “professional” C-channel.

Four audio interface receivers (IIS1, IIS3, and the two EBU receivers) send their received data on an internal 40-bit wide bus, the Internal Audio Data Bus. Every transmitter sources its data to be transmitted from this same internal bus. Every transmitter has a multiplexer to select the data source. Possible sources are (IIS1 receiver, IIS3 receiver, two EBU receivers, processor data output1, processor data output2, processor data output 3). Every transmitter also has a FIFO after the multiplexer. This FIFO gives the data source some freedom when data is generated. The FIFOs compensate for phase shifts when a transmitter takes data from another receiver. In the case that the transmitter sends out processor-generated data, the FIFO allows the processor to send several audio words in one burst to the audio transmitter.

To allow the processor to receive and transmit audio data, an interface is present between the internal Audio Data Bus and the ColdFire memory space. As shown in Figure 17-1, this interface is seen in the memory map as Processor Data Interface Registers. Three of these are Processor Data Out registers, PDOR1, PDOR2 and PDOR3. When the processor writes to one of these registers, the data is sent directly to the Internal Audio Data Bus, and depending on the setting of the multiplexers (**13, 15,** and **24)** it will end up in one or several of the transmit FIFOs **(12, 14,** and **25)**. There are three Processor Data In registers, PDIR1, PDIR2, and PDIR3. When the processor reads from one of these address locations, it actually reads data from one of the FIFOs (**17**, **17a, or 17b**). These FIFOs receive data from the Internal Audio Data Bus using multiplexers **(16**, **16a, and 16b)**. Depending on the setting of the multiplexers, data from one of the audio data receivers will end in the FIFOs. Possible receivers for the three PDIR channels are IIS1 receiver, IIS3 receiver and the two IEC958 receivers.

Besides the mechanism to let the processor access the audio data, there are several interrupts and control registers to allow the processor to determine when it should read or write data to the appropriate Processor Data Interface Register.

The IEC958 receiver and transmitter handle the main data audio stream in the same way as the IIS receivers and transmitters. This is done using the internal Audio Data Bus. Additionally, they support the IEC958 “C” and “U” channels. IEC958 “C” and “U” channel data is interfaced directly to memory-mapped registers (**22, 26, 27** and **28**).

17.2 Audio Interface Memory Map and Register Definitions

The registers and register field descriptions are provided in this section for the Audio Interface. These registers include the following registers: Interrupt Mask and Interrupt Status registers, Serial and Digital Audio Interface registers, Receive Interface registers, Data Exchange registers, CDROM Block Encoder and Decoder registers, DMA Configuration registers, Phase Configuration register, and the XTrim register.

17.3 Audio Interface Memory Map

All of the Audio Interface registers listed in Table 17-1 have already been shown in the various parts of this chapter. They are repeated here as a quick reference.

Table 17-1. Audio Interface Memory Map

Address	Register Name	Sze Bits	Access
MBAR2 + 0x12	I^2S1 configuration	32	RW
MBAR2 + 0x16	I^2S 2 configuration	32	RW
MBAR2 + 0x1A	I^2S 3 configuration	32	RW
MBAR2 + 0x1E	I^2S 4 configuration for SCLK4	32	RW
MBAR2 + 0x20	EBU 1 configuration	32	RW
MBAR2 + 0x20	EBU 2 configuration	32	RW
MBAR2 + 0x24	EBU 1 Receive C Channel	32	R
MBAR2 + 0xD0	EBU 2 Receive C Channel	32	R
MBAR2 + 0x28	EBU 1 Transmit C Channel	32	RW
MBAR2 + 0x2C	EBU 2 Transmit C Channel	32	RW
MBAR2 + 0x32	DataIn Control	16	RW
MBAR2 + 0x34 MBAR2 + 0x38 MBAR2 + 0x3C MBAR2 + 0x40	Processor data in 1 Left (PDIR1-L)	32	R
MBAR2 + 0x44 MBAR2 + 0x48 MBAR2 + 0x4C MBAR2 + 0x50	Processor data in 3 Left (PDIR3-L)	32	R
MBAR2 + 0x54 MBAR2 + 0x58 MBAR2 + 0x5C MBAR2 + 0x60	Processor data in 1 Right (PDIR1-R)	32	R

Table 17-1. Audio Interface Memory Map (continued)

Address	Register Name	Sze Bits	Access
MBAR2 + 0x64 MBAR2 + 0x68 MBAR2 + 0x6C MBAR2 + 0x70	Processor data in 3 Right (PDIR3-R)		
MBAR2 + 0x34 MBAR2 + 0x38 MBAR2 + 0x3C MBAR2 + 0x40	Processor data out 1 Left (PDOR1-L)	32	W
MBAR2 + 0x44 MBAR2 + 0x48 MBAR2 + 0x4C MBAR2 + 0x50	Processor data out 1 Right (PDOR1-R)	32	W
MBAR2 + 0x54 MBAR2 + 0x58 MBAR2 + 0x5C MBAR2 + 0x60	Processor data out 2 - Left (PDOR2-L)	32	W
MBAR2 + 0x64 MBAR2 + 0x68 MBAR2 + 0x6C MBAR2 + 0x70	Processor data out 2 - Right (PDOR2-R)	32	W
MBAR2 + 0x74 MBAR2 + 0x78 MBAR2 + 0x7C MBAR2 + 0x80	Processor data out 3 left + right (PDOR3)	32	W
MBAR2 + 0x74 MBAR2 + 0x78 MBAR2 + 0x7C MBAR2 + 0x80	Processor data in 2 left + right (PDIR2)	32	R
MBAR2 + 0x84	U Channel Transmit	32	RW
MBAR2 + 0x88	U Channel Receive	32	R
MBAR2 + 0x8C	Q Channel Receive	32	R
MBAR2 + 0x92	CD Text Control	16	RW
MBAR2 + 0x9F	DMA Configure	8	RW
MBAR2 + 0xA2	Phase Configure	8	RW
MBAR2 + 0xA6	XTRIM	16	RW
MBAR2 + 0xA8	Frequency measurement	32	R
MBAR2 + 0xC8	Block decoder/encoder control	32	RW
MBAR2 + 0xCE	audioGlob	16	RW

17.4 Audio Interrupt Mask and Status Register Descriptions

The interrupts of the audio interface use vectors 0–31 of the interrupt controller. There are two sets of registers associated with interrupt operation.

Table 17-2. Interrupt Register Addresses

Address	Name	Width	Description	Reset Value	Access
MBAR2 + 0x94 MBAR2 + 0x97	InterruptEn	32	Interrupt enable register	0	R/W
MBAR2 + 0x98 MBAR2 + 0x9B	InterruptStat	32	Interrupt status register	–	R
MBAR2 + 0x9 MBAR2 + 0x9B	InterruptClear	32	Interrupt clear register	–	W
MBAR2 + 0xE4 MBAR2 + 0xE7	InterruptEn3	32	Interrupt enable register	–	R/W
MBAR2 + 0xE0 MBAR2 + 0xE3	InterruptStat3	32	Interrupt status register	–	R
MBAR2 + 0xE0 MBAR2 + 0xE3	InterruptClear3	32	Interrupt clear register	–	W

Every pending audio interrupt will show up as a ‘1’ in register InterruptStat or InterruptStat3. The interrupt will cause the associated interrupt to go active if the corresponding bit in InterruptEn is set to ‘1‘. Most interrupts are cleared by writing a ‘1’ to the corresponding bit in InterruptClear register.

Table 17-3. Interrupt Register Description

Bit	Interrupt Name	Description	Vector	How to Clear
31	IIS1TXUNOV	I^2S 1 transmit FIFO under/over	31	reg. IntClear
30	IIS1TXRESYN	I^2S 1 transmit FIFO resync	30	reg. IntClear
29	IIS2TXUNOV	I^2S 2 transmit FIFO under/over	29	reg. IntClear
28	IIS2TXRESYN	I^2S 2 transmit FIFO resync	28	reg. IntClear
27	EBUTXUNOV	IEC958 transmit FIFO under/over	27	reg. IntClear
26	EBUTXRESYN	IEC958 transmit FIFO resync	26	reg. IntClear
25	EBU1CNEW	IEC958-1 receiver new C channel received	25	reg. IntClear
24	EBU1VALNOGOOD	IEC958-1 receiver validity bit not set	24	reg. IntClear
23	EBU1SYMERR	IEC958-1 receiver symbol error	23	reg. IntClear
22	EBU1BITERR	IEC958-1 receiver parity bit error	23	reg. IntClear
21	UCHANTXEMPTY	U Channel transmit register is empty	21	write to tx reg
20	UCHANTXUNDER	U Channel transmit register underrun	20	reg. IntClear
19	UCHANTX NEXTFIRST	U Channel transmit register next byte will be first	19	write to Tx reg
18	U1CHANRCVFULL	U1Channel receive register full	18	read Rcv reg

Table 17-3. Interrupt Register Description (continued)

Bit	Interrupt Name	Description	Vector	How to Clear
17	U1CHANRCVOVER	U1Channel receive register overrun	23	reg. IntClear
16	Q1CHANRCVFULL	Q1Channel receive register full	18	read rcv reg
15	Q1CHANRCVOVER	Q1Channel receive register overrun	13	reg. IntClear
14	UQ1CHANSYNC	U/Q Channel sync found	18	reg. IntClear
13	UQ1CHANERR	U/Q Channel framing error	13	reg IntClear
12	PDIR1UNOV	Processor data in 1 under/over	12	reg IntClear
11	PDIR1RESYN	Processor data in 1 resync	11	reg IntClear
10	PDIR2UNOV	Processor data in 2 under/over	10	reg IntClear
9	PDIR2RESYN	Processor data in 2 resync	9	reg IntClear
8	AUDIOTICK	*Tick* interrupt	8	reg IntClear
7	U2CHANRCVOVER Q2CHANOVERRUN UQ2CHANERR	IEC 958 receiver 2 U/Q channel error	7	reg IntClear
6	PDIR3 RESYNC	Processor data in 3 resync	6	reg IntClear
5	PDIR3 FULL	Processor data in 3 full	5	read fromPDIR3
4	IIS1TXEMPTY	I^2S 1 transmit FIFO empty	4	write to FIFO
3	IIS2TXEMPTY	I^2S 2 transmit FIFO empty	3	write to FIFO
2	EBUTXEMPTY	EBU transmit FIFO empty	2	write to FIFO
1	PDIR2 FULL	Processor data in 2 full	1	read from PDIR2
0	PDIR1 FULL	Processor data in 1 full	0	read from PDIR1

Table 17-4. InterruptEn3 InterruptClear3, InterruptStat3 Register Description

Bit	Interrupt Name	Description	Vector	How to Clear
25	EBU2CNEW	IEC958-2 receiver new C channel received	17	reg. IntClear3
24	EBU2VALNOGOOD	IEC958-2 receiver validity bit not set	16	reg. IntClear3
23	EBU2SYMERR	IEC958-2 receiver symbol error	15	reg. IntClear3
22	EBU2BITERR	IEC958-2 receiver parity bit error	15	reg. IntClear3
18	UCHANRCVFULL	U2 Channel receive register full	14	read rcv reg
17	UCHANRCVOVER	U2 Channel receive register overrun	7	reg. IntClear3
16	QCHANRVFULL	Q2 Channel receive register full	14	read rcv reg
15	QCHANOVERRUN	Q2 Channel receive register overrun	7	reg. IntClear3
14	UQCHANSYNC	U/Q2 Channel sync found	14	reg. IntClear3
13	UQCHANERR	U/Q2 Channel framing error	7	reg IntClear3

17.5 Serial Audio Interface (I^2S/EIAJ) Register Descriptions

There are a total of three serial audio interfaces. Each interface can handle Philips I^2S or Sony EIAJ protocol. Interface 1 is a receive/transmit interface. Interface 2 is transmit only, Interface 3 is a receive only. Every serial audio interface block has a 32-bit configuration register associated with it.

NOTE

Each of the three I^2S interfaces is capable of operating in Philips I^2S mode or Sony EIAJ mode with either 32, 36, or 40 bits per word clock. Timing diagrams describing each of these modes are given in the following sections. The frequency of the clock and data signals is programmable, as is the inversion of the bit clock (SCLK) or word clock (LRCK) for each I^2S interface.

Inversion of the LRCK clock only operates correctly on a slave receiver, therefore IIS3. If IIS1 is being used for transmit and receive in master mode then LRCK will be inverted on both the input and the output. Thereby cancelling the effect.

The SCLK and LRCK signals for each I^2S interface can either be inputs to the interface or they can be generated internally (outputs). See Table 17-5.

Figure 17-2 illustrates the valid bits in the IIS1 Configuration Registers and Table 17-5 provides the description of the bit fields.

Address MBAR2 + 0x10 (reset 0x0fc8) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R														EF/CFLG INSERT	CFLG SAMPLE POSITION	TXSOURCE SELECT
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	CLOCKSEL				TX FIFO CONTROL	TXSOURCE SELECT			SIZE		MODE	LRCK FREQUENCY			LRCK INVERT	SCLK INVERT
W																
Reset	0	0	0	0	1	1	1	1	1	1	0	0	1	0	0	0

Figure 17-2. IIS1 Configuration Registers (0x10)

Figure 17-3 illustrates the valid bits in the IIS2 Configuration Registers and Table 17-5 provides the description of the bit fields.

Address MBAR2 + 0x14 (reset 0x0fc8) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																TXSOURCE SELECT
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	CLOCKSEL				TX FIFO CONTROL	TXSOURCE SELECT			SIZE		MODE	LRCK FREQUENCY			LRCK INVERT	SCLK INVERT
W																
Reset	0	0	0	0	1	1	1	1	1	1	0	0	1	0	0	0

Figure 17-3. IIS2 Configuration Registers (0x14)

Figure 17-4 illustrates the valid bits in the IIS3 and IIS4 (SCLK4) Configuration Registers and Table 17-5 provides the description of the bit fields.

Address MBAR2 + 0x18 (reset 0x0fc8) (IIS3config)
MBAR2 + 0x1C (reset 0x0fc8) (IIS4config(SCLK4)) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	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	CLOCKSEL								SIZE		MODE	LRCK FREQUENCY			LRCK INVERT	SCLK INVERT
W																
Reset	0	0	0	0	1	1	1	1	1	1	0	0	1	0	0	0

Figure 17-4. IIS3 and IIS4 (SCLK4) Configuration Registers (0x18, 0x1C)

Table 17-5. IIS Configuration Registers Field Descriptions

Field	Description
EF/CFLG	EF/CFLG insert. See note 16 0 Not active 1 Active
CFLG	CFLG sample position. See note 16 0 Sample CFLG input 1 SCLK clock after incoming LRCK edge 1 Sample CFLG input 6 SCLK clocks before incoming LRCK edge

Table 17-5. IIS Configuration Registers Field Descriptions (continued)

Field	Description
CLOCKSEL	See notes 1, 11, 14 and 17 following bit these descriptions. 0000 SCLK/LRCK is input 0001 SCLK: Audio Clk/ 24 0010 SCLK: Audio Clk / 16 0011 SCLK: Audio Clk / 12 0100 SCLK: Audio Clk / 8 0101 SCLK: Audio Clk / 6 0110 SCLK: Audio Clk / 4 0111 SCLK: Audio Clk / 3 1100 SCLK: Audio Clk / 2 1000 SCLK, LRCK: follow IIS1 1001 SCLK, LRCK: follow IIS2 1010 SCLK, LRCK: follow IIS3 1011 Reserved
TX FIFO CONTROL	See notes 2, 7, 13, and 15 following these bit descriptions. 1 Reset to 1 sample remaining 0 Normal operation
TXSOURCE SELECT	See notes 2, 9, 12, and 15 following bit these descriptions. 0 000 Digital zero 0 001 PDOR1 0 010 PDOR2 0 011 PDOR3 0 100 IIS1 RcvData 0 101 IIS3 RcvData 0 110 Reserved 0 111 EBU RcvData 1 000 EBU2 RcvData
SIZE	See notes 3, 4, and 8 following bit these descriptions. 00 16 bits 01 18 bits 10 20 bits 11 Zero
MODE	1 Sony, EIAJ mode 0 Philips I^2S mode
LRCK FREQUENCY	100 64 bit clocks / word clock 010 48 bit clocks / word clock 000 32 bit clocks / word clock Other settings: reserved, undefined
LRCK INVERT	See note 5 following bit these descriptions. 1 Invert on word clock 0 No invert on word clock

Table 17-5. IIS Configuration Registers Field Descriptions (continued)

Field	Description
SCLK INVERT	See note 6 following bit these descriptions. 1 Invert on bit clock 0 No invert on bit clock

[1] Audio Clk is typically 11.2896 MHz or 16.93 MHz. Actual value given Table 4-4 in Chapter 4, “Phase-Locked Loop and Clock Dividers.”

[2] When bit 11 is set, FIFO is in reset condition. The FIFO is always re-set to “1 sample remaining”. The value of the remaining one sample will be all-zero.

[3] When Philips I^2S mode is selected, 16-18-20 bits will yield the same result.

[4] Internal interface is 40 bits / sample (20 left + 20 right). 16, 18 bit words are padded with zeros

[5] LRCK “invert” will invert the incoming LRCK signal between the pin and the serial data receiver and transmitter

[6] SCLK “invert” will invert the incoming SCLK signal between the pin and the serial data receiver and transmitter.

[7] Reset to one sample remaining is used to synchronize the data transfer from one input interface to another output interface running at the same frequency.

[8] “Zero” means data is transferred at the sampling frequency, with all data cleared down to digital zero.

[9] PDOR1, PDOR2, PDOR3: audio data output registers.

[10] Serial data transmit / receive interfaces have no limit on minimum incoming or outgoing sampling frequency. The maximum SCLK frequency is limited to 1/3 of the internal system clock (CPUclk/2). Mark/space ratio should be equal or better than 38/62.

[11] Reprogramming bits 15-12 during functional operation is not allowed. Reprogramming is only allowed while FIFO is in reset condition (bit 11 set ‘1’)

[12] When “digital zero” is selected as the source, the FIFO outputs “zero” on its outgoing data bus, regardless of the input side and content of the FIFO. No FIFO related exceptions are generated.

[13] When the FIFO leaves the reset state, because the user writes a “normal operation” state into the control register, the **FIFO is kept in reset until the first long-word is written to it.** As a result, the “start” of the normal operation is synchronized with the writing of the first data into the FIFO.

[14] When IIS/Sony interface LRCK/SCLK is set in “follow IIS” mode, the bit clock and word clock become exactly identical to bit and word clock of the “followed” interface. If e.g. LRCK/SCLK for IIS interface 2 is set in “follow IIS1”, the DAC or ADC connected to IIS2 can use the bit clock and word clock of IIS1. Note:- Bit and word clock for IIS2 can be used then used as GPIO if desired.

[15] Bit 16 extends the Tx FIFO control bit and the bit order becomes 16, 10, 9, 8.

[16] These bits should be programmed to zero for normal operation. For IIS1 receiver, it is possible to use the special EF/CFLG insertion mode, by setting bit 18 = 1. This mode is intended to interface with Philips CD decoders (SAA7324 and successors). When this mode is used, IIS1CONFIG must be programmed to “Sony” mode, 16 bits. The SAA7324 must also be programmed to “Sony” mode, 16 bits. The CFLG flag coming from SAA7324 must be connected with CFLG input. The EF flag coming from SAA7324 must be connected with EF input. If all this is done correctly, the device will receive the 16 MSB ‘s of the incoming data in bits [17:2] of the received serial data. Bit [1] of the received data is the EF flag of the corresponding word, as output by SAA7324. Bit [1] will be set if the MSB or the LSB or both are flagged. Bit [0] of the received data is the CFLG flag of the corresponding word, as output by SAA7324. These flags can be used for implementing an electronic shock protection FIFO.

[17] For IIS4 only the SCLK4 setting can be used. See Chapter 12, “Analog to Digital Converter (ADC)” for the purpose of this function.

17.5.1 IIS/EIAJ Transmitter Descriptions

The two I^2S/EIAJ transmitters operate independently. Each of the transmitters has the capability of transmitting data from one of several sources:

- One of the three processor data out registers.

- One of the two I^2S receivers.
- The digital audio (EBU) receiver.
- Digital zero.

The source of the transmit data is programmable.

17.5.2 IIS/EIAJ Transmitter Interrupts

There are a number of interrupts defined for use with the serial audio transmitters:

- Serial audio interface1 transmit FIFO overrun or underrun
- Serial audio interface1 transmit FIFO left/right resynchronization
- Serial audio interface1 transmit FIFO empty
- Serial audio interface 2 transmit FIFO overrun or underrun
- Serial audio interface 2 transmit FIFO left/right resynchronization
- Serial audio interface 2 transmit FIFO empty

The action of the IIS transmitters on FIFO underrun is to repeat the last sample.

Timing diagrams for IIS/EIAJ mode are shown in Figure 17-5 and Figure 17-6. Data and word clock output is clocked on the falling edge of the SCLK bit clock (noninverted).

17.5.3 IIS/EIAJ Receiver Descriptions

Each of the two IIS receivers operate independently. For timing diagrams, see Figure 17-5 and Figure 17-6. The data can be clocked into each receiver using an external or internally-generated SCLK/LRCK. Data is always clocked on the rising edge of the SCLK bit clock (non-inverted).

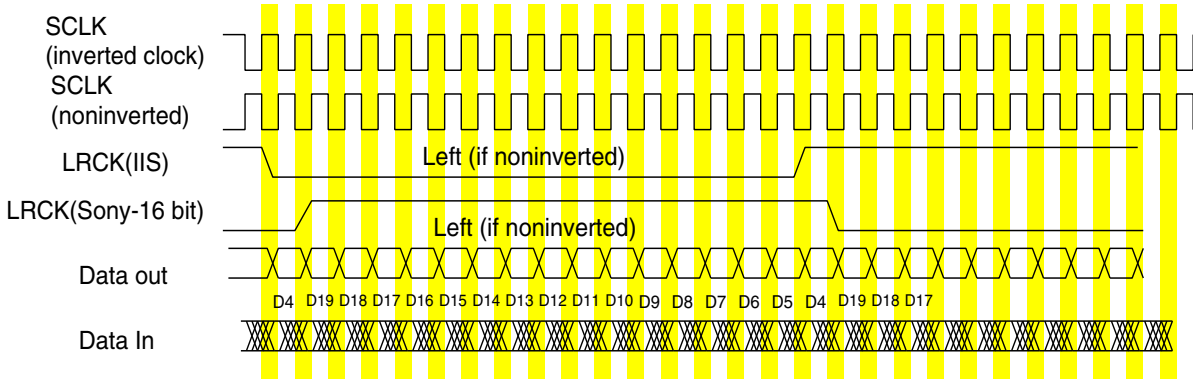


Figure 17-5. IIS/EIAJ Timing Diagram (16 SCLK edges per word)

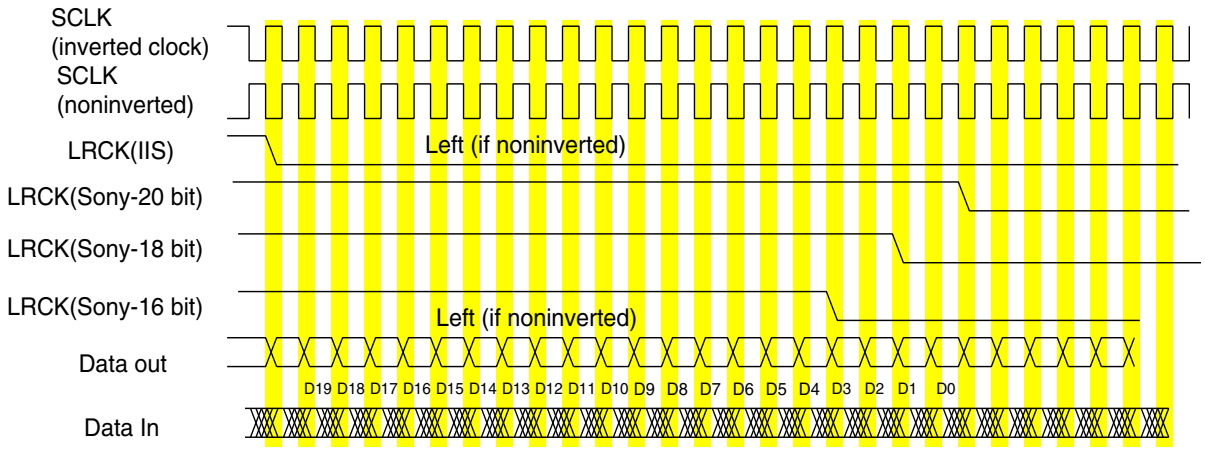


Figure 17-6. IIS/EIAJ Timing Diagram (24 or 32 SCLK edges per word)

NOTE

In 18-bit mode, bits D1 and D0 are set 0.
In 16-bit mode, bits D3,D2,D1 and D0 are set 0.

17.6 Digital Audio Interface (EBU/SPDIF) Register Descriptions

Figure 17-7 illustrates the valid bits in the EBU1Config Registers and Table 17-6 provides the descriptions of the bit fields.

Address MBAR2 + 0x20 / 0x24 (Reset 0x3F00) Access: User read/write

	16	15–12	11	10–8	7–6	5	4–2	1–0
R W	TXSOURCE SELECT	CLOCKSEL	TX FIFO CONTROL	TXSOURCE SELECT	IEC958 RECEIVE SOURC SELECT	VAL CONTROL	IEC958 OUT SELECT	U SOURCE SELECT
Reset	0	0 0 0 0	0	0 0 0	0 0	0	0 0 0	0 0

Figure 17-7. EBU1Config Register

Table 17-6. EBU1Config Register Field Descriptions

Field	Description	Reset	Notes
15–12 CLOCKSEL	0000 IEC958 clock: audioclk / 16 0001 IEC958 clock: audioclk / 12 0010 IEC958 clock: audioclk / 8 0011 IEC958 clock: audioclk / 6 0100 IEC958 clock: audioclk / 4 0101 IEC958 clock: audioclk / 3 0110 IEC958 clock: sclk1 0111 IEC958 clock: sclk2 1000 IEC958 clock: sclk3 1001 IEC958 clock: sclk4	0011	1,2,8
11 TX FIFO CONTROL	0 Normal operation 1 Reset to one sample remaining	1111	3,5,11

Table 17-6. EBU1Config Register Field Descriptions (continued)

Field	Description	Reset	Notes
16,10–8 TXSOURCE SELECT	0 000 Digital zero 0 001 PDOR1 0 010 PDOR2 0 011 PDOR3 0 100 iis1RcvData 0 101 iis3RcvData 0 110 Reserved 0 111 ebu1RcvData 1 000 ebu2RcvData	1111	3,6,9
7–6 IEC958 RECEIVE SOURCE SELECT	00 EBU in 1 01 EBU in 2 10 EBU in 3 11 EBU in 4	00	
5 VALCONTROL	0 Outgoing V flag always 1 1 Outgoing V flag always 0	0	10
4–2 IEC958 OUT SELECT	000 Off. Output 0 001 Feed-through EBUIn1 010 Feed-through EBUIn2 011 Feed-through EBUIn3 100 Feed-through EBUIn4 101 Normal operation	000	12

Table 17-6. EBU1Config Register Field Descriptions (continued)

Field	Description	Reset	Notes
1–0 U SOURCE SELECT	00 No embedded U channel 01 U channel from IEC958 receive block. (CD mode) 10 Reserved, undefined 11 U channel from on-chip U channel transmitter.	00	4

1 The IEC958 interface needs 64 * audio sample frequency clock for good operation. This is 2.822 MHz for operation at a sample rate of 44.1 kHz.

2 When The IEC958 transmitter is set to follow SCLK1, SCLK2, SCLK3, or SCLK4, it will transmit at the same rate as the serial audio interface only if the interface uses 64 bit clocks / word clock format.

3 When bit 11 is set, the FIFO is in its reset condition. The FIFO is always re-set to “contain 1 sample”. This sample value is re-set at the same time to “all-zeros”.

4 U channel selection is described on section handling subcode processing.

5 Before starting IEC958 transmission to copy data from another incoming channel, first reset the FIFO to one sample remaining, while the source selector is set to correct source. When the FIFO is switched to normal operation, transmission will start normally.

6 Digital zero means data transmitted is digital zero, while “C” and “U” channel contain valid data. When digital zero is transmitted, the IEC958 transmit FIFO is not read any more by the IEC958 transmit hardware.

7 PDOR1, PDOR2, PDOR3: Processor Data Out Register.

8 Reprogramming bits 15-12 during functional operation is not allowed. Reprogramming is only allowed while FIFO is in its reset condition (bit 11 set ‘1’)

9 When “digital zero” is selected as a source, the FIFO outputs “zero” on its outgoing data bus, regardless of the input side and content of the FIFO. No FIFO related exceptions are generated.

10 This bit controls the outgoing validity flag of the EBU transmitter. When it is re-set, all outgoing data is flagged as “valid”. If it is set, all data is flagged “invalid”.

11 When the FIFO leaves the reset state, because the user write a “normal operation” state into the control register, the **FIFO is kept into reset until first long-word is written to it.** As a result, the “start” of the normal operation is synchronized with the writing of the first data into the FIFO.

12 This field selects what is output on EBUOUT1. If the field is “000,” the SPDIF output is off and outputs 0. If the field is “001” to “100,” it muxes out one of the EBUIN’s to the EBUOUT, without any reformatting. When the field is set to “101,” this is normal operation of the SPDIF transmitter.

Figure 17-8 illustrates the valid bits in the EBU2Config Register and Table 17-7 provides the description of the bit fields.

Address MBAR2 + 0XD0: 0XD3 (RESET 0X3F00) Access: User read/write

	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R										IEC958 RECEIVE SOUCE SELECT							
W																	
Reset	0	0	0	1	1	1	1	1	1	0	0	00	0	0	0	0	0

Figure 17-8. EBU2Config Register

Table 17-7. EBU2Config Register Field Descriptions

Field	Description	Reset
7–6	IEC958 Receive source select. 00 EBU in 1 01 EBU in 2 10 EBU in 3 11 EBU in 4	00

17.6.1 IEC958 Receive Interface

The IEC958 (SPDIF) receive interface consists of 2 blocks:

1. The source selector
2. The IEC958 receiver itself

The source is selected by programming the appropriate EBU Control Register bits 7:6. The receiver then extracts the data from the stream and outputs the data on the internal audio bus. The data can then be used by the processor (using the PDIR and other registers) or by the IIS or EBU transmit interface. In the case of the data being used as input to one of the IIS transmitters, the data rate of the incoming EBU data must match exactly with that of the IIS transmitter. The following functions are performed by the block.

17.6.1.1 Audio Data Reception

The IEC958 receive block **(19)** extracts the audio data from the stream and puts this in 20-bit format on the Internal Audio Data bus. The format is exactly the same as the format produced by the serial data interfaces.

17.6.1.2 Control Channel Reception Register Descriptions

There are two 32-bit registers, one for each receiver, which receive the first 32 bits of the “C” channel. No interpretation is done. For a description of the control (or “C”) channel in EBU data formatting, refer to the IEC958-3 specification’s description of control channel. Figure 17-9 illustrates the valid bits in the EBURcvCChannel.

Address EBU1RCVCCHANNEL MBAR2 + 0X24
EBU2RCVCCHANNEL MBAR2 + 0XD4

Access: User read only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	EBURcvC Channel1 and Channel2															
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	EBURcvC Channel1 and Channel2															
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 17-9. EBURcvCChannel Register

Bits are ordered first bit left. So, C-channel bit “0” is seen in bit position 31 in the EBURcvCChannel register. C-channel bit “31” is seen as the LSB bit in the register.

17.6.1.3 Control Channel Interrupt (IEC958 “C” Channel New Frame)

When the value of a new IEC958 “C” channel frame is loaded into the EBURcvCChannel register, an interrupt is generated. This interrupt is cleared when the processor writes the corresponding bit in the InterruptClear register. EBURcvCChannel is double buffered. However the register can be read at any time and provide true values the interrupt only indicates that a NEW “C” channel value has been loaded.

17.6.1.4 Validity Flag Reception

An interrupt is associated with the Validity flag. (interrupt 24 - IEC958ValNoGood). This interrupt is set every time a frame is seen on the IEC958 interface with the validity bit set to “invalid”.

17.6.1.5 IEC958 Exception Definition

There are several IEC958 exceptions defined that will trigger an interrupt. These are:

- Control channel change—Set when EBURcvCChannel register is updated. The register is updated for every new C-Channel received. The exception is reset when EBURcvCChannel register is read.
- EBU Illegal Symbol—Set on reception of illegal symbol during IEC958 receive. Reset by writing register InterruptClear. Refer to Section 17.7.7, “Audio Interrupts” for details. The EBU input is a biphase/mark modulated signal. The time between any two successive transitions of the EBU signal is always 1, 2 or 3 EBU symbol periods long. The EBU receiver will parse the stream, and split it in so-called symbols. It recognizes s1, s2 and s3 symbols, depending on the length of the symbols. Not all sequences of these symbols are allowed. To give an example, a sequence s2-s1-s1-s1-s2 cannot occur in a error-free EBU signal. If the receiver finds such an illegal sequence, the *illegal symbol* interrupt is set. No corrective action is undertaken.

  When the interrupt occurs, this means:

  a) The EBU signal is has been affected by noise

b) The EBU frequency has changed

- IEC958 bit error—Set on reception of bit error. (Parity bit does not match). Reset on write to InterruptClear register. Refer to Section 17.7.7, “Audio Interrupts” for details.

17.6.1.6 EBU Extracted Clock

The clock from the EBU signal is extracted for measurement purposes only. It cannot be used as a clock to drive other audio interfaces like IIS. The average rate is 128 x the sampling frequency (ex. 128 * 44.1 KHz for 44.1 KHz input sampling frequency). The internal signal is used by the FreqMeas circuit (and with suitable software) to calculate the incoming sample rate. It can also be used to calculate the offset between the incoming SPDIF audio clock and the audio clock input at CRIN. This offset value can then be used to calculate the necessary trim required to have the CRIN clock locked to the incoming SPDIF clock. This is achieved via suitable external hardware and the XTRIM pin. In this way we can provide a inherently stable and jitter free SPDIF locked clock for the rest of the application. The resultant audio clock jitter produced is then solely a result of the stability of the crystal used as the CRIN clock source.

17.6.1.7 Reception of User Channel and CD-Subcode Over IEC958 Receiver

The IEC958 receiver is capable of extracting the User Channel bits out of the data stream. The extracted bits are assembled in the 32-bit UChannelReceive register, with the first U-Channel bit in the MSB position (bit 31). The interface can be configured to detect Sync patterns in the U-Channel in the case the U-Channel contains CD subcode (CD-mode). The Sync Detection can be enabled by setting the USyncMode bits in the CD-Subcode register (Table 17-11). Sync recognition is done as follows:

- Internally, a symbol starting with a “1” is treated as a “data symbol”. Any consecutive 11 zeros are treated as a “zero symbol”.
- The sync detector will assume User Channel sync whenever:
  (a) A sequence of 4 symbols, data-sync-sync-data, is found.
  (b) 98 symbols (does not matter data or zero) after the previous “sync symbols”.
- The ChannelLengthError interrupt is set when a new sync is not found at the correct distance from the previous sync, or if UChannelReceive or QChannelReceive do not contain the correct number of bits/bytes.

Furthermore, in CD-mode, the Q-channel receiver extracts the Q-channel CD-Subcode from the U-Channel stream and assembles the bits in the 32-bit “QChannelReceive” with the first bit in the MSB position.

17.6.1.8 U Channel Receive and Q Channel Receive Register Descriptions

Figure 17-10 illustrates the valid bits in the U Channel Receive and Q Channel Receive Registers and Table 17-8 provides the description of the bit fields.

Address MBAR2 + 0x88: 0x8B (U channel 1)
MBAR2 + 0xD8: 0xDB (U channel 2)
MBAR2 + 0x8C: 0x8F (Q channel 1)
MBAR2 + 0xDC: 0xDF (Q channel 2)

Access: User read only

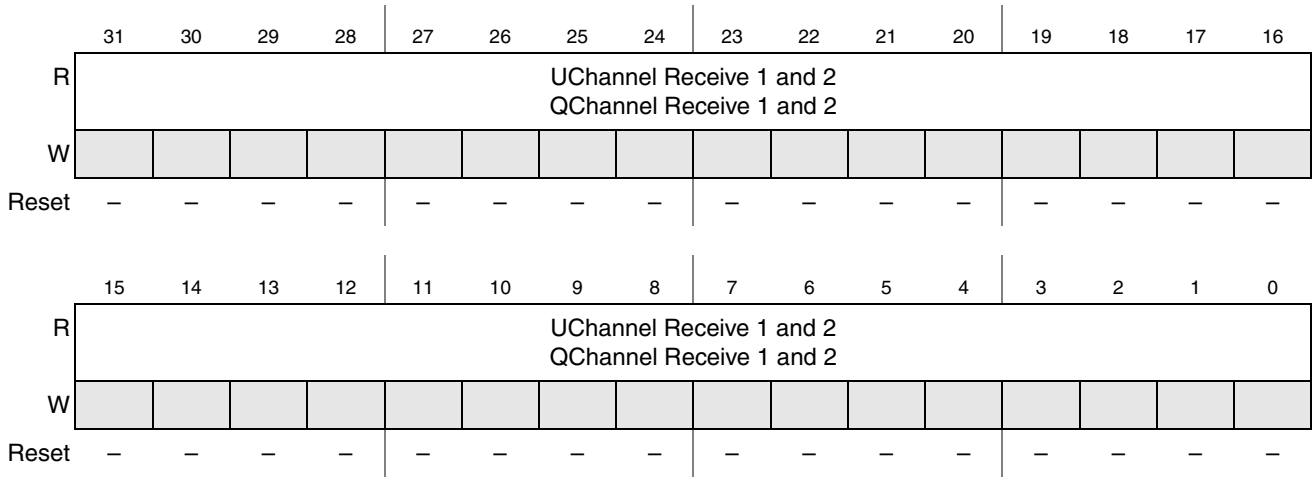


Figure 17-10. U Channel Receive and Q Channel Receive Registers

Table 17-8. U Channel Receive and Q Channel Receive Registers Field Descriptions

Field	Description
31–0 UCHANNEL RECEIVE 1 AND 2	U channel receive register. Contains next 4 U channel bytes.
31–0 QCHANNEL RECEIVE 1 AND 2	Q channel receive register. Contains next 4 Q channel bytes.

Figure 17-11 illustrates the valid bits in the CDTEXTCONTROL Register and Table 17-9 provides the description of the bit fields.

Address MBAR2 + 0x92

Access: User read/write

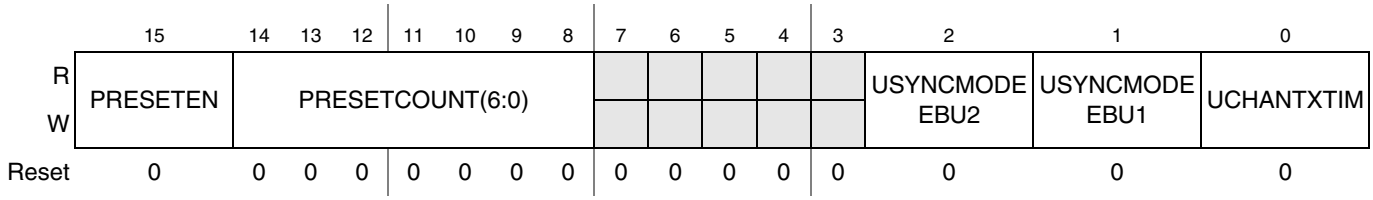


Figure 17-11. CDTEXTCONTROL Register

Table 17-9. CDTEXTCONTROL Register Field Descriptions

Field	Description	Notes
15 PRESETEN	0 No action on free-running sync position counter 1 Preset free-running sync position counter	2, 3
14–8 PRESETCOUNT	Sync presetting count	1, 3

Table 17-9. CDTEXTCONTROL Register Field Descriptions (continued)

Field	Description	Notes
7–3	Reserved.	–
2 USYNCMODE EBU2	0 Other data 1 CD user channel reception	–
1 USYNCMODE EBU1	0 Other data 1 CD user channel reception	–
0 UCHANTXTIM	0 Timing to reg. *UChannelTx* from cd-text output interface 1 Timing to reg. *UChannelTx* from EBU1 output interface	–

[1] On read back, last written value is returned.
[2] On read back, zero is returned.
[3] PRESETCOUNT*(6:0)* will only affect the free running counter when the register is written with PRESETEN = '1'. Writing with PRESETEN = '0' does not affect the counter.

17.6.1.9 U and Q Receive Register Interrupts

- UChannelRcvFull—Receive register full
- UChannelRcvOverrun—Overrun error
- QChannelRcvFull—Receive register full
- QChannelOverrun—Overrun error on Q channel
- ChannelSyncFound—Received sync on U/Q channel.
- ChannelLengthError—Set when ChannelSyncFound occurs when there are less than 32 bits waiting in QchannelReceive register, or less than 4 bytes in UChannelReceive, or when a syncing error is found. To regain correct syncing, U channel receive register and Q channel receive register must be read to establish correct synchronization.

On the input interface, 2 data receive registers are defined:

1. UChannelReceive—32-bit register to receive U-channel incoming subcode.
2. QChannelReceive—32-bit register to receive Q-channel of incoming subcode.

The hardware associated with the IEC958 receiver U-channel reception is intended for reception of the following kind of data:

- CD or CD-compatible User channel subcode (P,Q and R-W, or Q and R-W). See the CD Red Book specification for a detailed description.
- Other types of subcode.

17.6.1.10 Behavior of User Channel Receive Interface (CD Data)

This section details the behavior of the user channel receive interface on incoming CD user channel subcode in the IEC958 receiver. This mode is selected if UsyncMode (bit 1) in register CD-Subcode control, is set.

The CD subcode stream embedded into the IEC958 User channel consists of a sequence of packets. Every packet contains 98 symbols. The first two symbols of every packet are sync symbols and the other 96 symbols are data symbols.

Any sequence found in the IEC958 U-channel stream starting with a leading one, followed by 7 information bits, is recognized as a data symbol. Subsequent data symbols are separated by pauses. During the pause, zero bits are seen on the IEC958 U-channel.

Data symbols come in MSB first. The MSB is the leading one and is always received as bit 7.

When a long pause is seen between 2 subsequent data symbols, the IEC958 receiver assumes the reception of one or more sync symbols. Table 17-10 shows this functionality.

Table 17-10. Correlation Between Zero Bits and Sync Symbols

No of U Channel Zero Bits	Corresponding Number of Sync Symbols
0–1	Unpredictable, not allowed
2–10	0
11–22	1
23–34	2
35–45	3
> 45	Unpredictable, not allowed

The recognition of the number of sync symbols derives from the fact that the U-channel transmitter in the CD channel decoder will transmit one symbol on average every 12 IEC958 channel bits. On this average rate, there is a tolerance of 5% maximum.

The IEC958 receiver is tolerant on symbol error. Due to the physical nature of the transmission of the data over the CD disc, not more than one out of any 5 consecutive user channel symbols may be in error. The error may cause a change in data value, which is not treated by this interface, or it may cause a data symbol to be seen as a sync symbol, or a sync symbol to be seen as a data symbol. However, not more than one out of any 5 consecutive user channel symbols can be affected in this way.

The IEC958 User channel circuitry will recognize the 98-symbol packet structure. The 96 symbol payload is transferred using 2 registers as follows:

- The UChannelRcv register—In this register, data is presented 4 symbols at a time. Every time 4 new valid symbols, received on the IEC958 U-Channel, are present, the UChannelRcvFull interrupt is asserted. For one 98-symbol packet, 96 symbols are carried across UChannelRcv. To transfer all this data, 24 UChannelRcvFull interrupts are generated.
- The QChannelRcv register—In this register, only the Q bit of the packet is accumulated. Operation is similar to UChannelRcv. Because only Q-bit is transferred, only 96 Q-bits are transferred for any 98-symbol packet. To transfer this data, 3 QChannelRcvFull interrupts are generated. When QChannelRcvFull occurs, it is coincident with UChannelRcvFull. There is only one QChannelRcvFull for every 8 UChannelRcvFull. The convention is that the most significant data is transmitted first, and is left-aligned in the registers.

The timing, as it applies to packet boundary, is extracted by hardware.The last UChannelRcvFull corresponding to a given packet should be coincident with the last QChannelRcvFull. In this last U, Q channel interrupt, symbols 95-98 are received, as are Q-channel bits 67-98. The interrupts are coincident with ChannelSyncFound, flagging the last symbols of the current frame.

When the start of a new packet is found before the current packet is complete (less than 98 symbols in the packet), the ChannelLengthError interrupt is set. The application software should read out UChannelRcv and QchannelRcv registers, discard the value, and assume the start of a new packet.

As previously mentioned, packet sync extraction is tolerant for single-symbol errors. Packet sync detection is based on the recognition of the sequence data-sync-sync-data in the symbol stream, because this is the only syncing sequence that is not affected by single errors. If the sync symbol is not found 98 symbols after the previous occurrence, it is assumed to be destroyed by channel error, and a new sync symbol is interpolated.

Normally, only data bytes are passed to the application software. Every data byte will have its most significant bit set. If sync symbols are passed to the application software (i.e., the processor), they are seen as all-zero symbols. Sync symbols can only end up in the data stream due to channel error.

17.6.1.11 Behavior of User Channel Receive Interface (non-CD data)

This section details the behavior of the user channel receive interface on incoming non-CD data.

This mode is selected if UsyncMode (bit 1) in register CD Text control is set ‘0’.

In non-CD mode, the IEC958 User channel stream is recognized as a sequence of data symbols. No packet recognition is done.

Any sequence found in the IEC958 U-channel stream starting with a leading one, followed by 7 information bits, is recognized as a data symbol. Subsequent data symbols are separated by pauses. During the pause, zero bits are seen on the IEC958 U-channel.

Four consecutive data symbols seen in the IEC958 U-Channel stream are grouped together into the UChannelRcv register. First symbol is left, last symbol is right aligned. Whenever UChannelRcv contains 4 new data symbols, UChannelRcvFull is asserted.

In this mode, the operation of QchannelRcv and associated interrupt QChannelRcvFull is reserved, undefined. Also reserved, undefined is the operation of ChannelLengthError and ChannelSyncFound.

The U-channel is extracted and output by the IEC958 Receive block on EBURcvUChannelStream. Processing is done by the CD-Subcode as described in Section 17.7, “Processor Interface Overview.”

17.6.2 IEC958 (SPDIF) Transmit Interface

The IEC958 interface provides the necessary features to allow transmitting of digital data according to the IEC958 specification with the exception that only 20-bit data is supported. The 4 LSB’s of the 24-bit data word are always ‘0’. In addition to data, the interface allows for transmission of the C- and U-channels and control over the Valid flag. Note: For the U-channel, only the CD User Data format is supported.

NOTE

EBUOUT1 will output a clock signal just after reset and before they can be configured as GPIO. The frequency of the clock output will be CRIN/16.

17.6.2.1 Transmit “C” Channel

The “C” channel includes control bits such as data valid, copy protected, transmitted sample rate etc.

Table 17-11. EBU1TxCChannel Registers Addresses

Address	Name	Width	Description	Reset Value	Access
MBAR2 + 0x28	EBU1TxC Channel1	32	“C” channel bit settings for IEC958 transmitter - Consumer format	Undefined	RW

Table 17-12. Formatting of EBUOUT1 (Consumer “C” Channel)

IEC958 Bits[1]	Field Name	Description	Taken From
0–31	CONTROL	Relevant data	EBU1TXCCHANNEL1(31:0)
32–191	–	–	Always 0

[1] Ordering of bits in Ebu1TxCChannel1 is MSB sent out first. So, Ebu1TxCChannel1(31) is sent out as IEC958 bit 0.

17.6.2.2 IEC958 Transmitter Interrupt Conditions

There are three transmitter interrupt conditions:

1. Transmit FIFO underrun
2. Transmit FIFO overrun
3. Transmit FIFO empty

17.6.2.3 IEC958-3 Ed2 and Tech 3250-E Standards Compliance

The IEC958 transmitter is compliant with IEC958-3 Ed2 and Tech 3250-E documents from international IEC standards committee and European Broadcasting Union organizations.

The IEC958 transmitter implementation allows any sample frequency. Operation is guaranteed up to a maximum incoming transmit clock of 16MHz. The mark/space ratio of the transmit clock must be equal to or better than 38/62.

17.6.2.4 Transmission of U-Channel and CD Subcode Data

The user channel transmitter is intended to assemble the CD subcode stream, and conform to the IEC958 CD standard specification. The generation of the data needs to be done in software and loaded into hardware registers. The Audio peripheral has provisions to insert this CD subcode stream into the outgoing IEC958 stream, or to transmit it over a dedicated 3-wire interface, called the CD-Subcode interface. The 3-wire CD-Subcode is intended to connect Philips Semiconductor CD encoder devices.

This combined interface provides output formats for both CD-Subcode and IEC958 U-channel. The same data is used for both output formats.

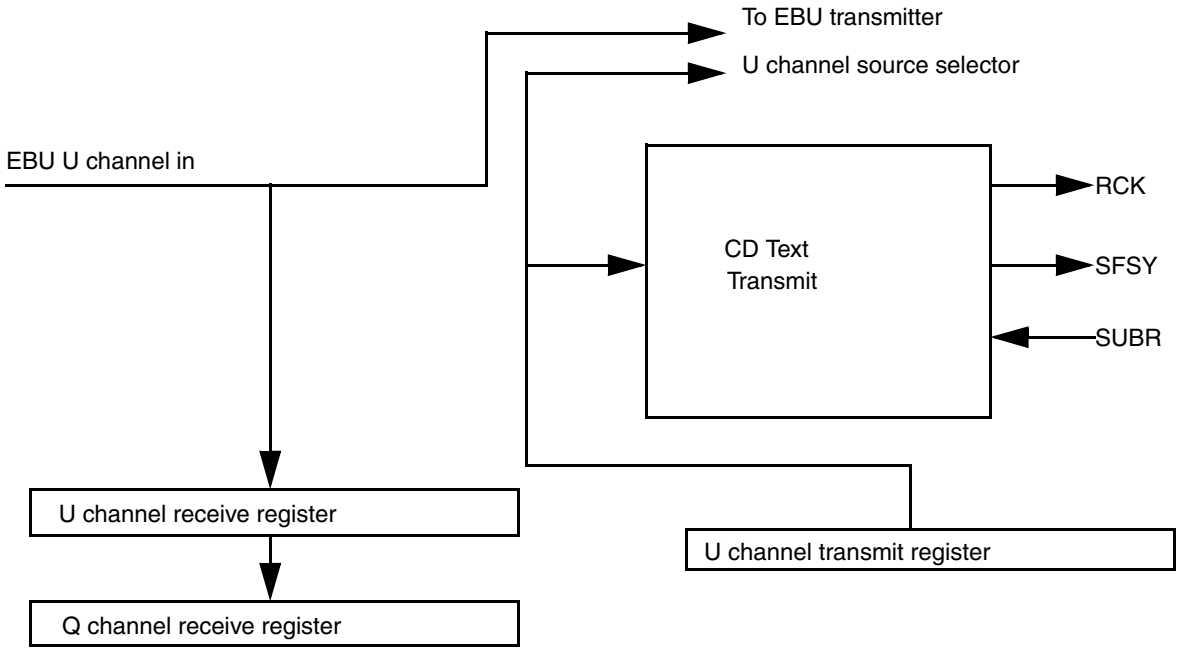


Figure 17-12. CD-Subcode Interface

Table 17-13. UChannel Transmit Register

Address	Name	Width	Description	Reset Value	Access
MBAR2 + 0x84	UChannel Transmit	32	U channel transmit register. Contains next 4 U channel bytes.	–	RW

Address MBAR2 + 0x92 Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PRESET EN	PRESET COUNT(6:0)												Usync mode EBU1	Usync mode EBU1	UChan TxTim
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-13. CD-Subcode Register

17.6.3 CD Subcode Interrupts

The following interrupts are associated with the CD Subcode data:

- UChannelTxEmpty—Register is empty, needs re-loaded.
- UChannelTxUnderrun—Under run error on register.
- UChannelTxNextFirstByte—Received indication from CD-Subcode output interface that next word to be written contains first byte of the 96-byte U-channel frame.CD Subcode Interface: SFSY, RCK and SUBR.

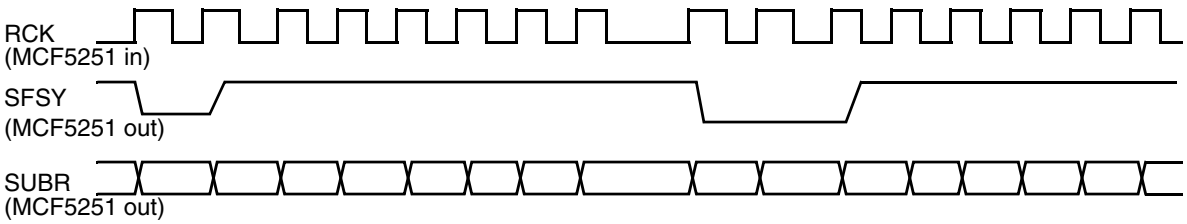


Figure 17-14. Data Format on CD-Subcode Interface Out

RCK is the incoming clock from the channel encoder. SFSY is used to flag the first symbol bit, and first packet symbol. During the first bit of every symbol, SFSY is low, During the first two bits of the first symbol of every packet, SFSY is low.

SUBR is the data out, used to transmit outgoing data in serial form. The most significant bit is transmitted first.

RCK is an input, SFSY and SUBR are outputs.

CD User channel subcode is transmitted out of the 3-wire CD subcode interface. This user channel subcode needs to be assembled by the ColdFire processor application software.

The CD-Subcode format has a 98-symbol packet structure. Of these 98 packets, the first 2 symbols are sync symbols, followed by 96 8-bit data symbols.

The boundaries of the 98-symbol packets are determined by free-run counters. The first symbol of any packet is transmitted with the special sync sequence on SFSY. The first and second symbols are all-0 symbols. The other 96 symbols need to be uploaded by the application software in register UChannelTransmit.

Upload is done by application software handshaking to interrupt UChannelTxEmpty. If this interrupt is set, the application software uploads 4 symbols of the current user channel packets into register UChannelTx.

The interrupt UChannelTxNextFirstByte flags the start of a new U-channel packet. It is always coincident with UChannelTxEmpty, and signal that the first 4 symbols of a new packet need to be loaded into UChannelTx.

The following pseudo-code reacts on both interrupts. One interrupt handler can take care of both UchannelTxEmpty and UChannelTxNextFirstByte. This last interrupt is not enabled.

```
if(UChannelTxEmpty interrupt) then
  if(UChannelTxNextFirstByt interrupt set also) then
    reset this interrupt
    synchronize pointer to sent out new frame
  end if ;
  load UChannelTransmit with data from pointer
  update pointer
  reset interrupt
end if ;
```

17.6.3.1 Free Running Counter Synchronization

There is a synchronization issue on start-up between the MCF5251 and some channel encoders. On start-up, the RCK clock is kept silent. At a certain point in time, the CDR60[1] will start clocking the RCK,

and then it will require that the first symbol transmitted from the MCF5251 to the CDR60 is a sync symbol. If this is not the case, the CDR60 fails to synchronize.

To solve the synchronization issue, the counter that determines the sync position can be preset using the register CdTextControl (Table 17-11).

17.6.3.2 Controlling the SFSY Sync Position

When RCK is not clocking, it is possible to control the subcode byte number that will be sent out next by the CD-Subcode interface by writing CdTextControl with PresetEn set to 1.

- When 0 is written to presetCount, the next byte sent out is a CD-Subcode sync byte. (SFSY low).
- When a value (97–i) is written to presetCount, i non-sync bytes are transmitted, followed by a sync byte.
- After writing to CdTextControl with PresetEn set to 1, next bit out is always the first bit of a new byte.
- Writing CdTextControl with PresetEn set to 1, while RCK is running, will result in unpredictable, undefined operation.

17.6.4 Inserting CD User Channel Data Into IEC958 Transmit Data

Source selection of data transmitted into the User Channel of the IEC958 transmitter is selected by bits (1,0) of register EBUConfig.

- When selected the source is the IEC958 receiver, every user channel data byte received into the input of the IEC958 user channel, is inserted into the outgoing stream at approximately. the same time it was found in the incoming stream.
- When the selected source is CD-Subcode, every data byte transmitted over the CD-Subcode output is also inserted into the IEC958 output stream. The most significant bit of every byte is transmitted as a “1”. All sync symbols are transmitted as all-0.
- In case the RCK clock is not present, it is still possible to use the CD-Subcode interface to assemble the outgoing IEC958 User channel data. In this case, bit UChanTxTim in register CDTEXTCONTROL must be set ‘1’ (Table 17-11). It will cause the timing to the CD-Subcode registers to be controlled by the IEC958 transmitter. One symbol (data or sync) will be transmitted into the IEC958 output every 12 User Channel data bits.

17.7 Processor Interface Overview

The interface between the processor and the Audio Modules is given in this section. Figure 17-15, shows a simplified picture of the interface between the audio modules and the processor core.

NOTE

The audio module register addresses are relative to the MBAR2 register.

1. CDR60 is the informal name for Philips CD-R channel encoder

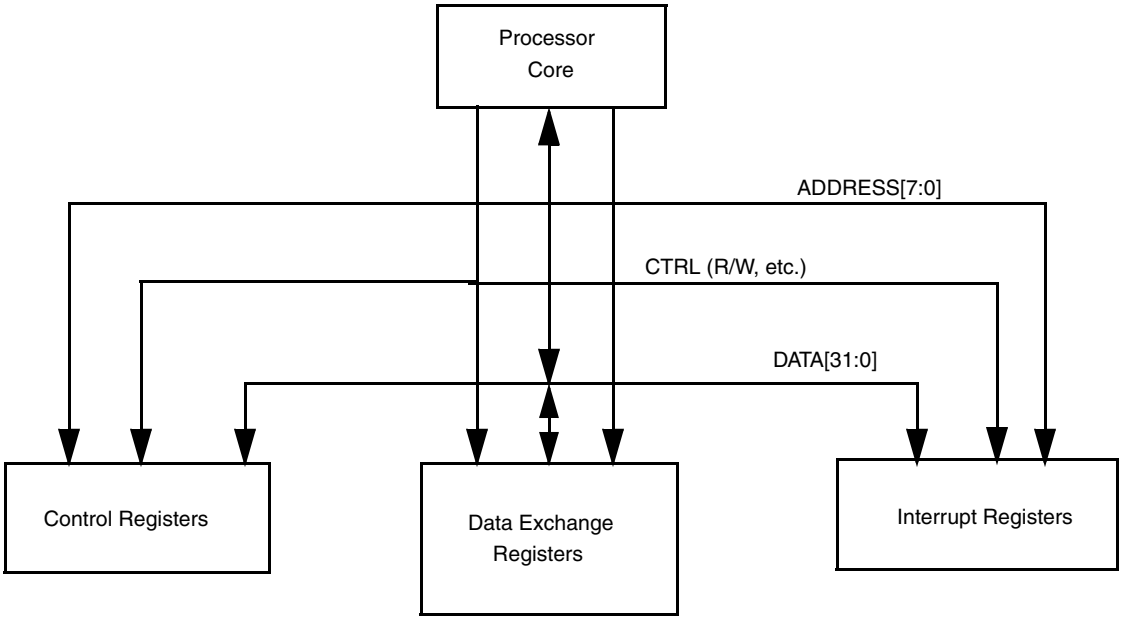


Figure 17-15. Processor/Audio Module Interface

17.7.1 Data Exchange Register Descriptions

Table 17-14 shows the Data Exchange Registers. To read/write data to/from the audio modules, use the registers as shown in Table 17-1.

Table 17-14. Data Exchange Register Descriptions

Address MBAR2 +	Name	Width	Description[1, 2]	Reset Value	Access
0x34 0x38 0x3C 0x40	PDIR1-L	32	Processor data in Left. Multiple address to read this register allows MOVEM instruction to read FIFO.	–	R
0x44 0x48 0x4C 0x50	PDIR3-L	32	Processor data in Left. Multiple address to read this register allows MOVEM instruction to read FIFO.	–	R
0x54 0x58 0x5C 0x60	PDIR1-R	32	Processor data in Right Multiple address to read this register allows MOVEM instruction to read FIFO.	–	R
0x64 0x68 0x6C 0x70	PDIR3-R	32	Processor data in Right Multiple address to read this register allows MOVEM instruction to read FIFO.	–	R
0x34 0x38 0x3C 0x40	PDOR1-L	32	Processor data out 1 Left. Multiple address to write this register allows MOVEM instruction to write FIFO.	undef	W

Table 17-14. Data Exchange Register Descriptions (continued)

Address MBAR2 +	Name	Width	Description[1, 2]	Reset Value	Access
0x44 0x48 0x4C 0x50	PDOR1-R	32	Processor data out 1 Right Multiple address to write this register allows MOVEM instruction to write FIFO	undef	W
0x54 0x58 0x5C 0x60	PDOR2-L	32	Processor data out 2 Left Multiple address to write this register allows MOVEM instruction to write FIFO	undef	W
0x64 0x68 0x6C 0x70	PDOR2-R	32	Processor data out 2 Right Multiple address to write this register allows MOVEM instruction to write FIFO	undef	W
0x74 0x78 0x7C 0x80	PDOR3	32	Processor data out 3 left + right	undef	W
0x74 0x78 0x7C 0x80	PDIR2	32	Processor data in 3 left + right	undef	R

1 Multiple addresses for PDOR/PDIR fields are intended for easy use of MOVEM instruction to move data into and out of the FIFOs. The data read at each address of any range is exactly the same, being the next sample in/out of the FIFO. There is no difference in FIFO operation between a read at address e.g. 0x74, 0x78, 0x7C.

2 There are memory overlaps between PDIR's and PDOR's. PDOR's cannot be read, PDIR cannot be written.

17.7.2 Data Exchange Register Overview

- PDOR1-L, PDOR1-R: (Processor Data Out 1). These are 32-bit registers. Both registers have 4 consecutive longword addresses assigned (multiple decode). This allows easy transfer of multiple samples using MOVEM instructions. Data written to these registers will end in one of the FIFO 's (Figure 17-1) **12, 14, 17, 17a, 17b** or **25**. The format of data in the registers is defined below.
- PDOR2-L, PDOR2-R: (Processor Data Out 2). Same function as PDOR1. Both (PDOR2-L and PDOR2-R) registers occupy 4 consecutive longword addresses (multiple decoded.) Data written to it will end in one of the FIFO 's. (fig. 17-1) **12,14,17, 17a, 17b** or **25**.
- PDOR3: (Processor Data Out3). Same function as PDOR1. But it is a single 32-bit register which contains both Left + Right data in 16-bit precision occupying 4 consecutive longword addresses. Data written to it will end in one of the FIFO 's. (fig 17-1) **12,14, 17a, 17b** or **25**.
- PDIR1-L, PDIR1-R (Processor data in). Used to transfer data to the processor. These 32-bit registers, each occupy 4 consecutive longword addresses are used to read data from the audio bus. Data flowing in is selected by source multiplexer **16a**. Control via register DataInControl (12, 2:0). Table 17-15.

- PDIR2 (Processor data in). Same function as PDIR1. Single 32-bit register contains both Left + Right in 16-bit precision. Data flowing in is selected by source multiplexer **16**. Control via register *DataInControl(13,5:3)* Table 17-15.
- PDIR3-L, PDIR3-R (Processor data in). This function is identical to PDIR1. Data flowing in is selected by source multiplexer **16b**. Control via register *DataInControl(19:16)*. Table 17-15.

17.7.2.1 Data In Selection

The DataInControl register determines what data will be in the PDIR1 input FIFO, in PDIR2 input FIFO, and in the PDIR3 input FIFO. All FIFO's are six-deep, and have programmable "full" indication.

Address MBAR2 + 0X30 (RESET 0X00) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R									PDIR3 ZERO CTRL	PDIR3 RESET	PDIR3 FULL INTERRUPT		SELECT PDIR3			
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	PDIR2 FULL INTERRUPT SELECT		SELECT PDIR2	SELECT PDIR1	PDIR2 ZERO CTRL	PDIR1 ZERO CTRL	PDIR2 RESET	PDIR1 RESET	PDIR1 FULL INTERRUPT SELECT		SELECT PDIR2		SELECT PDIR1			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-16. DataInControl Register

NOTE

The DataInControl register bits 7:6 allow selection when FIFO full flag is set. This is necessary due to polling. It may be necessary to service the FIFO when it is less than completely full. For PDIR2 only, interrupt-driven and DMA-driven read-out is supported.

Table 17-15. DataInControl Register Field Descriptions

Field	Description	Reset
31–24	Reserved.	0
23 PDIR3 ZERO CONTROL	0 Normal operation 1 Always read zero from PDIR3	0
22 PDIR3 RESET	0 Normal operation 1 Reset PDIR3 to one sample remaining	0
21–20 PDIR3 FULL INTERRUPT SELECT	00 Full interrupt if at least 1 sample in FIFO 01 Full interrupt if at least 2 samples in FIFO 10 Full interrupt if at least 3 samples in FIFO 11 Full interrupt if at least 6 samples in FIFO	00

Table 17-15. DataInControl Register Field Descriptions (continued)

Field	Description	Reset
19–16 SELECT PDIR3	0000 Off 0001 PDOR1 0010 PDOR2 0011 Unused 0100 iis1RcvData 0101 iis3RcvData 0110 Reserved 0111 ebu1RcvData 1000 ebu2RcvData	000
15–14 PDIR2 FULL INTERRUPT SELECT	00 Full interrupt if at least 1 sample in FIFO 01 Full interrupt if at least 2 samples in FIFO 10 Full interrupt if at least 3 samples in FIFO 11 Full interrupt if at least 6 samples in FIFO	00
11 PDIR2 ZERO CONTROL	0 Normal operation 1 Always read zero from PDIR2	0
10 PDIR1 ZERO CONTROL	0 Normal operation 1 Always read zero from PDIR1	0
9 PDIR2 RESET	0 Normal operation 1 Reset PDIR2 to one sample remaining	0
8 PDIR1 RESET	0 Normal operation 1 Reset PDIR1 to one sample remaining	0
7–6 PDIR1 FULL INTERRUPT SELECT	00 Full interrupt if at least 1 sample in FIFO 01 Full interrupt if at least 2 samples in FIFO 10 Full interrupt if at least 3 samples in FIFO 11 Full interrupt if at least 6 samples in FIFO	00
13, 5–3 SELECT PDIR2	0 000 Off 0 001 PDOR1 0 010 PDOR2 0 011 Unused 0 100 iis1RcvData 0 101 iis3RcvData 0 110 Reserved 0 111 ebu1RcvData 1 000 ebu2RcvData	000
12, 2–0 SELECT PDIR1	0 000 Off 0 001 PDOR1 0 010 PDOR2 0 011 Unused 0 100 iis1RcvData 0 101 iis3RcvData 0 110 Reserved 0 111 ebu1RcvData 1 000 ebu2RcvData	000

17.7.3 PDIR and PDOR Field Formatting

Each PDIR, PDOR 32-bit register contains only 20 relevant data bits. Formatting is done as follows:

Table 17-16. PDIR1-L, PDIR3-L, PDOR1-L, PDOR2-L Formatting

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
N/A	N/A	L19	L18	L17	L16	L15	L14	L13	L12	L11	L10	L9	L8	L7	L6
Bit 15	**Bit 14**	**Bit 13**	**Bit 12**	**Bit 11**	**Bit 10**	**Bit 9**	**Bit 8**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
L5	L4	L3	L2	L1	L0	0	0	0	0	0	0	0	0	0	0

Table 17-17. PDIR1-R, PDIR3-R, PDOR1-R, PDOR2-R Formatting

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
n/a	n/a	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6
Bit 15	**Bit 14**	**Bit 13**	**Bit 12**	**Bit 11**	**Bit 10**	**Bit 9**	**Bit 8**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
R5	R4	R3	R2	R1	R0	0	0	0	0	0	0	0	0	0	0

Table 17-18. PDIR2, PDOR3 Formatting

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
L19	L18	L17	L16	L15	L14	L13	L12	L11	L10	L9	L8	L7	L6	L5	L4
Bit 15	**Bit 14**	**Bit 13**	**Bit 12**	**Bit 11**	**Bit 10**	**Bit 9**	**Bit 8**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4

1 L18 is bit 18 of left sample, ~L19 is inverse of bit 19 of left sample, R18 is bit 18 of right sample.

2 If incoming/outgoing interface use 16, 18 bits, data is aligned at the MSB side. LSB ‘s D1-D0 or D3-D0 will read all-zero. Written values are disregarded.

3 PDOR3, PDIR2 use only 16 MSB of both left and right.

4 Inversion of MSB ‘s L19 and R19 translates the format from 2-complement to unsigned. (The continuous range e.g. -0x8000 to +7FFF is translated to 0 to +0xFFFF)

17.7.4 Overrun and Underrun with PDIR and PDOR Registers

All PDOR and PDIR registers have different FIFOs for left and right channels. As a result, there is always the possibility that the left and right FIFOs may go out of sync due to FIFO underruns and FIFO overruns that affect only one part (left or right) of any FIFO. To prevent this from happening, two hardware mechanisms are available:

1. If PDIR1, PDIR2, or PDIR3 FIFO overrun occurs on, as an example, the right half of the FIFO, the sample that caused the overrun is not written to the right half (due to overrun). Special hardware will make sure the next sample is not written to the left half of the FIFO. If the overrun occurs on the left half of the FIFO, the next sample is not written to the right half of the FIFO.

2. If IIS1 or IIS2 Tx FIFO, or EBU Tx FIFO underruns on, for example, the right half of the FIFO, no sample leaves that FIFO. (because it was already empty.) Special hardware ensures that the next sample read from the left FIFO will not leave the FIFO. (No read strobe is generated). If the underrun occurs on the left half of the FIFO, next read strobe to the right FIFO is blocked.

17.7.5 Automatic Resynchronization of FIFOs

An automatic FIFO resynchronization feature is available on the MCF5251. It can be enabled or disabled separately for every FIFO. If enabled, the hardware will check if the left and right FIFOs are in sync, and if not, it will set the filling pointer of the right FIFO to be equal to the filling pointer of the left FIFO.

The operation is shown in Figure 17-17. Every FIFO auto-resync controller has a state machine with three states:

1. Off
2. Stand-By
3. On

In the On state, the filling of the left FIFO is compared with the filling of right, and if they are not equal, right is made equal to left, and an interrupt is generated.

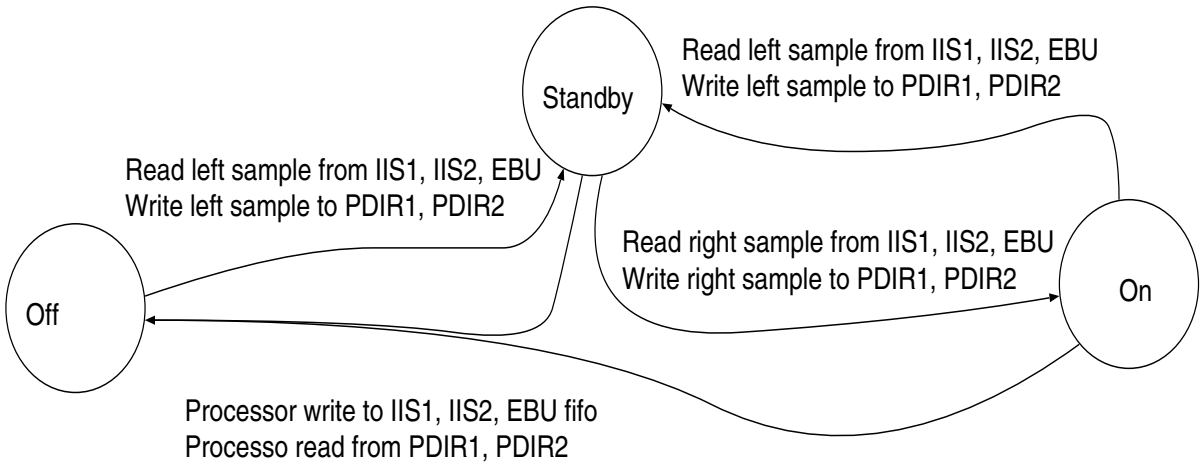


Figure 17-17. Automatic Resynchronization FSM of Left-Right FIFOs

The controller will stay in the Off state when the feature is disabled. When not disabled, the state machine will go to the Off state on any processor read or write to the FIFO. It will go from On or Off to Standby on any left sample read from IIS, IIS2, and EBU Tx FIFO's, or on any left sample write to PDIR1, PDIR2, PDIR3 FIFO's. The controller will go from Standby to On on any right sample read from IIS1, IIS2 and EBU Tx FIFO's, or on any right sample write to PDIR1, PDIR2 and PDIR3.

17.7.6 audioGlob Register Descriptions

Figure 17-18 illustrates the valid bits in the audioGlob Register and Table 17-19 provides the description of the bit fields.

Address 0xCC Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R				PDIR3 FIFO AUTO SYNC	AUDIOTICK SOURCE EBU2 EXT	EBU1 TX AUTO SYNC	IIS2 FIFO	AUTO SYNC	PDIR2 FIFO AUTO SYNC	PDIR1 FIFO AUTO SYNC	AUDIO_TICK COUNT		AUDIOTICK SOURCE			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-18. audioGlob Register

Table 17-19. audioGlob Register Field (0xCE) Descriptions

Field	Name	Description[1]	Reset	Notes
12	PDIR3 FIFO AUTO SYNC	0 Auto synchronization off 1 Auto synchronization on	0	
10	EBU TX AUTO SYNC	0 Auto synchronization off 1 Auto synchronization on	0	
9	IIS2 FIFO AUTO SYNC	0 Auto synchronization off 1 Auto synchronization on	0	
8	IIS1 FIFO AUTO SYNC	0 Auto synchronization off 1 Auto synchronization on	0	
7	PDIR2 FIFO AUTO SYNC	0 Auto synchronization off 1 Auto synchronization on	0	
6	PDIR1 FIFO AUTO SYNC	0 Auto synchronization off 1 Auto synchronization on	0	
5–3	AUDIO TICK COUNT	000 1 Interrupt for every event 001 2 Interrupt for every 2 events 010 3 011 4 100 5 Other Reserved, unused	000	
11, 2–0	AUDIO TICK SOURCE	0 000 Off 0 001 IIS1 Tx Right FIFO / Read 0 010 IIS2 Tx Right FIFO / Read 0 011 EBU Tx Right FIFO / Read 0 100 IIS1 Rcv Data 0 101 IIS3 Rcv Data 0 110 Reserved 0 111 EBU1 Rcv Data 1 000 EBU2 Rcv Data	000	

1 The automatic FIFO resynchronization can be switched on, and will avoid all mismatch between left and right FIFO's, if the software obeys following rules:

1. When left data is read or written to the left FIFO, in the same place of the program, data must be read or written to the right FIFO. Maximum time difference between left and right is 1/2 sample clock. (E.g. if the sample frequency is 44 kHz, then this is approximately 10 micro-seconds. For 88 kHz, then this approximately 5 micro-seconds.)
2. Writing or reading data to the FIFO 's must be at least 2 samples at the time. If there is a mis-match between Left-Right, the resync logic may go on only 1 sample clock after last data is read/written to the FIFO. Also acceptable is polling the FIFO, if at least part of the time, 2 samples will be read/written to it.

17.7.7 Audio Interrupts

17.7.7.1 AudioTick Interrupts

The audio tick interrupt is an interrupt to sustain an interrupt routine that is synchronous with one of the audio interfaces, but not directly related to any FIFO being full or empty. Two fields control how this interrupt is generated:

1. The source field controls the source event.
2. The count field controls the number of events (sample pairs) between any two audioTick interrupts.

For example, if the source is set to IIS1 Tx FIFO / Read, and count is set to three, the interrupt will occur after every three read strobes to the IIS1 Tx FIFO. Even if the FIFO is in reset state, the interrupt will continue running.

17.7.7.2 PDIR1, PDIR2, and PDIR3, Interrupts

With FIFO's feeding data to the PDIR registers, three interrupts are associated.

1. Full
2. Under/over
3. Resync

When the Full condition is set for processor data input registers, the processor should read data from the FIFO, before overrun occurs (this is within a 1/2 sample period). Reading of data should be done using 32-bit operands (ex. MOVE.L instruction). When the Full condition is set, and the FIFO contains, for example six samples, it is acceptable for the software to read the first six samples from the LEFT address, followed by six samples from the RIGHT address, or six samples from the RIGHT address, followed by six samples from the LEFT address, or one sample LEFT, followed by one sample RIGHT repeated six times. The order of reading does not need to be carried out in any specific order.

The implementation for PDIR1 is a double FIFO, one for left and one for right. The Full condition is set when both FIFOs are full. The Underrun/Overrun condition is set when one of the FIFO's actually underrun's or overrun's. The resync interrupt is set when the hardware took special action to resynchronize either the left or the right FIFO.

17.7.7.3 PDOR1, PDOR2, and PDOR3 Interrupts

Three interrupts are associated with FIFOs that can be written from PDOR1, PDOR2, PDOR3:

1. Empty
2. Under/over
3. Resync

When the Empty condition is set for processor data output registers, the processor should write data to the FIFO, before underrun occurs. Writing of data should be done using MOVE LONG or MOVEM instructions (with long-word oriented instructions). When Empty is set, and, for example, six samples need to be written, it is acceptable for the software to write first six samples from the LEFT address, followed

by six samples from the RIGHT address, or one sample LEFT, followed by one sample RIGHT repeated six times.

NOTE

In any chosen writing scheme the left should be written before the right.

The implementation of all data output FIFO's is a double FIFO, one for left and one for right. The Empty Interrupt is set when both FIFO's are empty. The Underrun/Overrun interrupt is set when one of the FIFO's either underrun's or overrun's. Resync is set when the hardware resynchronizes the left and right FIFOs.

On receiving an Underrun/Overrun interrupt, synchronization between Left and Right words in the FIFOs may be lost. Synchronization will not be lost when the underrun or overrun comes from the audio side of the FIFO. If the processor reads or writes more data from, for example, the left than from the right, synchronization will be lost. If automatic resynchronization is enabled, and if the software obeys the rules to let this work, resynchronization will be automatic.

Table 17-20. Interrupt Register Field Description (0x94, 0x98)

Bit	Interrupt Name	Description	How to Clear
31	IIS1TxUnOv	IIS1 transmit FIFO under/overrun	reg. IntClear
30	IIS1TxResyn	IIS1 transmit FIFO resync	reg. IntClear
29	IIS2TxUnOv	IIS2 transmit FIFO under/overrun	reg. IntClear
28	IIS2TxResyn	IIS2 transmit FIFO resync	reg. IntClear
27	EBUTxUnOv	EBU (IEC958) transmit FIFO under/overrun	reg. IntClear
26	EBUTxResyn	EBU (IEC958) transmit FIFO resync	reg. IntClear
25	EBUCNew	EBU (IEC958) Rx change of value of the C channel	reg. IntClear
24	IEC958ValNoGood	IEC958 Validity Flag no good	reg. IntClear
23	EBUSymErr	IEC958 receiver found illegal symbol	reg. IntClear
22	EBUBitErr	IEC958 receiver found parity bit error	reg. IntClear
21	UChanTxEm	UChannelTransmit register empty	write to tx reg
20	UChanTxUnder	UchannelTransmit register underrun	reg. IntClear
19	UChanTx-NextFirst	UchannelTransmit register next byte will be first	write to Tx reg
18	UChanRcvFull	UChannelReceive register full	read Rcv reg
17	UChanRcvOver	UChannelReceive register overrun	reg. IntClear
16	QChanRvFull	QChannelReceive register full	read rcv reg
15	QChanOverrun	QChannelReceive register overrun	reg. IntClear
14	UQChanSync	U/Q channel sync found	reg. IntClear
13	UQChanErr	U/Q channel framing error	reg IntClear
12	Pdir1UnOv	Processor data input underrun/overrun	reg IntClear
11	Pdir1Resyn	Processor data input resync	reg IntClear
10	Pdir2UnOv	Processor data input underrun/overrun	reg IntClear

Table 17-20. Interrupt Register Field Description (0x94, 0x98) (continued)

Bit	Interrupt Name	Description	How to Clear
9	Pdir2Resyn	Processor data input resync	reg IntClear
8	audioTick	audio tick interrupt	reg IntClear
7	U2CHANRCVOVER Q2CHANOVERRUN UQ2CHANERR	IEC958 receiver 2 U/Q channel error	reg IntClear
6	Pdir3Resyn	Processor data input resync	reg IntClear
5	PDIR3 full	Processor data input full	read from PDIR3
4	iis1TxEmpty	IIS 1 transmit FIFO empty	write to FIFO
3	iis2TxEmpty	IIS 2 transmit FIFO empty	write to FIFO
2	ebuTxEmpty	IEC958 transmit FIFO empty	write to FIFO
1	PDIR2 full	Processor data input full	read from PDIR2
0	PDIR1 full	Processor data input full	read from PDIR1

17.7.7.4 Audio Interrupt Routines and Timing

Usually, the processor will run an audio interrupt routine. Every time the audio interrupt routine runs, it will process 2, 3, or 4 audio samples, and send this many samples to one or more PDOR output registers. Also, the audio interrupt routine will read one or more PDIR registers until empty.

In the audio interrupt routine, typically at the beginning, the PDIR registers are read until empty, while the PDOR registers are written at the end of the routine when all calculations are completed. Due to this calculation latency, there is a delay between entering the audio interrupt routine and the filling of the transmit FIFOs.

Due to this delay, it is difficult to “fire” the audio interrupt routine on a transmit FIFO empty interrupt. Because of the extra delay before the data is written, the transmit FIFO will underrun before any data is written.

To make it easy for the programmer, the audioTick interrupt was added. To start the audio interrupt routine, use the following sequence:

1. Reset the transmit FIFOs
2. Program the transmit FIFOs to the correct source, then release the reset on transmit FIFOs
3. Reset the PDIR FIFOs
4. Load audio interrupt routine in on-chip SRAM
5. Release reset for the PDIR FIFOs and enable audioTick interrupt

The transmit FIFOs have a special feature. After the software releases the reset to them, they will stay in reset until the audio Interrupt Routine writes data to them for the first time. So, during Step 2 of above mentioned start-up procedure, all transmit FIFO’s are set in reset, with one sample remaining. They will stay in this state, until the audio Interrupt Routine writes data to them. At this point in time, they are then filled up with an extra 2, 3, or 4 samples to a total of 3, 4, or 5 samples. Also, the first data write to the

FIFOs releases the reset, and starts transmission of the FIFO data on the corresponding transmit output. (IIS1, IIS2 or IEC958). The next time that data is written to the FIFO's in the audioTick interrupt routine, 2,3, or 4 samples have been transmitted and the FIFO is ready to accept new data.

To work properly, the jitter from one audioTick write point to the next is important. Jitter should be lower than 1 sample period if data is written in groups of 2 or 3 samples to the transmit FIFOs, and lower than 1/2 sample period if data is written in groups of 4 samples to the transmit FIFOs.

The receive FIFO's (PDIR's) don't have an auto-reset-de-assert mechanism, and should be released out of reset just before enabling audioTick interrupt.

Figure 17-19 shows the timing (relative to the Word Clock) of the Empty, Under-run, and Audio Tick interrupts. Each FIFO holds up to six audio samples (left and right).

The Empty Interrupt occurs when there is still one right sample left to be transmitted, thus giving the system one audio sample length to fill the FIFO back-up. The Underrun Interrupt occurs when there are no samples left to be transmitted. While this is a situation that should be taken seriously, it will rarely occur, if at all. However, should this happen, the system will continue to repeat the last sample until the FIFO buffer has new data.

The Audio Tick Interrupt was introduced to aid a busy system by allowing the Interrupt to occur after a number of (programmable) sample pairs. In this example, the Audio Tick Interrupt has been set to trigger after the 4th sample pair. This gives the system up to two audio sample pairs to respond and fill the FIFO. This avoids the under-run issue. The decision to use the Audio Tick interrupt as apposed to the Empty Interrupt is dependent on the system and the reaction time of that system. Therefore, it is not expected that the Audio Tick Interrupt needs to be employed in all systems.

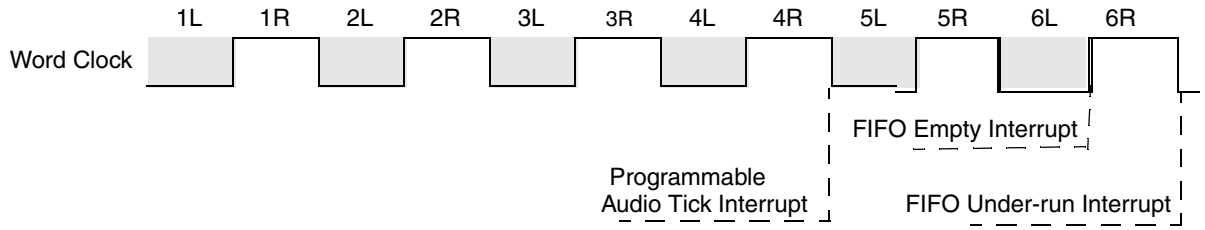


Figure 17-19. Audio Transmit / Receive FIFOs

17.7.8 CD-ROM Block Encoder and Decoder Register Descriptions

The processor interface registers PDOR3 and PDIR2 are equipped with a CD-ROM block encoder/decoder. The two interfaces are fully independent. One control register is associated with the interface.

Address MBAR2 BAS + 0xC8 Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DECODE SWAP		DECODE SYNC ALLOW		DECODE DE-SCRAMBLE	DECODE MODE			ENCODE SWAP		ENCODE SYNC ALLOW		ENCODE SCRAMBLE	ENCODE MODE		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-20. BlockControl Register

Table 17-21. BlockControl Register Field Descriptions

Bit Number	Bit Name	Description	Reset	Notes
15–14	DECODE SWAP	See note 1. Block decode swap control.	00	1
13	DECODE-SYNC ENABLE	See note 2. 1 Sync detection enabled. 0 Sync detection disabled	0	2
11	DECODE ENABLE	See note 3. 1 Descramble enabled. 0 Escramble disabled	0	3
9, 10	DECODE MODE	See note 4. 00 No CRC check 01 Mode 1 10 Mode 2, form 1 11 Mode 2, form 2	00	4
6, 7	ENCODE SWAP	See note 1. Block encode swap control	00	1
5	ENCODE SYNC ENABLE	See note 2. 1 Outgoing sync detecting enabled 0 Outgoing sync detecting disabled	0	2
3	ENCODE ENABLE	See note 3. 1 Scramble active 0 Scrambling inactive	0	3
1, 2	ENCODE MODE	00 No CRC instructed 01 Mode 1 10 Mode 2, form 1 11 Mode 2, form 2	00	4, 5

Notes:
1. See Table 17-22 for definition of how the swap is done.
2. Decode Sync Allow and Encode Sync Allow define whether the interfaces recognize the CD-ROM syncs embedded in the CD-ROM sectors. If this bit is switched on, then the interface will recognize the start of a new sector after finding the sync sequence in the data. If the bit is switched off, or if no sync sequence is found, sector start is assumed to be one sector length (2352 bytes) after the previous sector.
3. CDROM descrambling/scrambling control if required.
4. Mode selection determines how the CRC is calculated. The CRC depends on the CD-ROM mode/form, as defined in CD standards.
5. inserted CRC will over-write processor written data. (Processor sets CRC to any value, logic overwrites this.)

Table 17-22. Swap Control in CD-ROM Encoder/Decoder

Swap Field	Swap Action[1]
00	dataOut(31:0):= dataIn(31:0)
01	dataOut(31:16):= dataIn(15:0) dataOut(15:0):= dataIn(31:16)
10	dataOut(31:24):= dataIn(23:16) dataOut(23:16):= dataIn(31:24) dataOut(15:8):= dataIn(7:0) dataOut(7:0):= dataIn(15:8)
11	dataOut(31:24):= dataIn(7:0) dataOut(23:16):= dataIn(15:8) dataOut(15:8):= dataIn(23:16) dataOut(7:0):= dataIn(31:24)

1 Notation used is 32-bit words. Bits31-16 are part of the LEFT sample, bits 15-0 are part of the RIGHT sample.

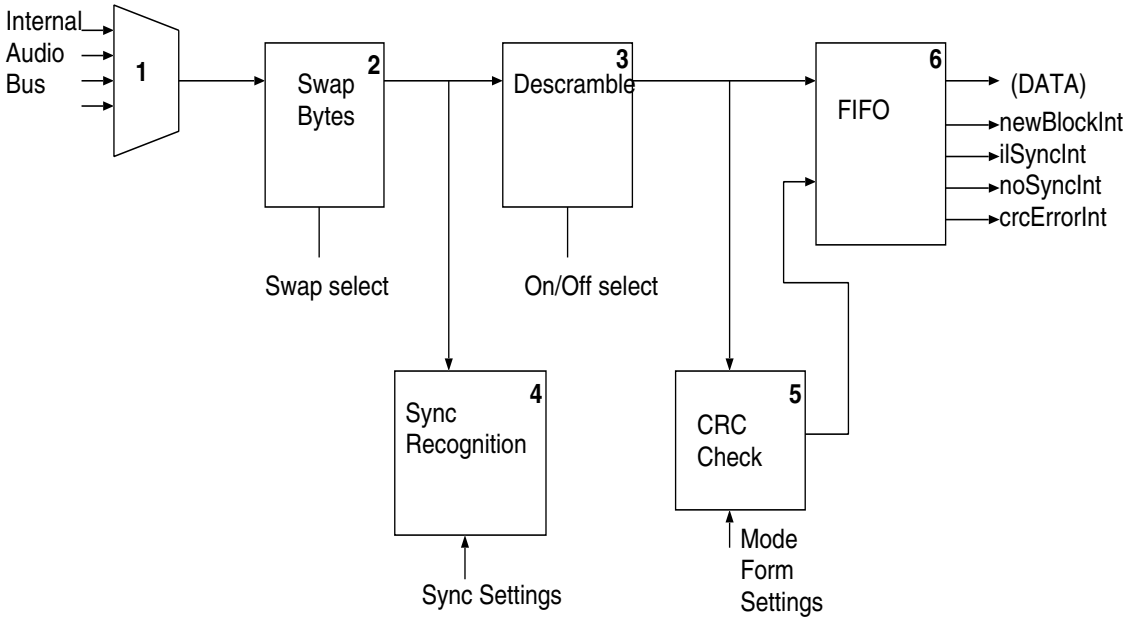


Figure 17-21. Block Decoder

17.7.8.1 CD-ROM Decoder Interrupts

The block decoder can detect and flag sync patterns and error conditions. The following conditions are flagged in status bits and each of these can generate an interrupt. All interrupts occur when the corresponding data word reaches the output of the FIFO.

- newBlock interrupt—Set when the next longword to be read is the first word of new block.
- noSync interrupt—Set when the next longword to be read is the first word of new block, and no valid sync pattern was found before the start of this new block in the stream.
- ilSync interrupt—Set when the next longword to be read is the first word of a new block, and the length of previous block was not equal to 2352 bytes (the nominal block length).

- crcError interrupt—Set when the next longword to be read is the first word of a new block, and CRC check on the previous block failed.

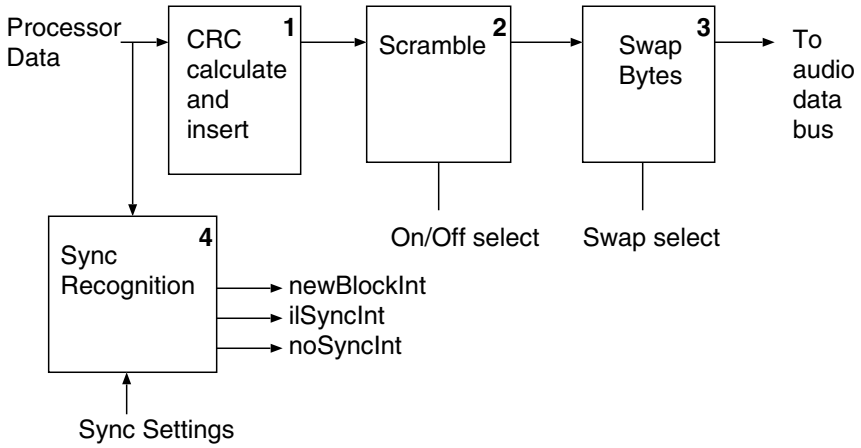


Figure 17-22. Block Encoder

The block encoder works on the incoming PDOR3 stream. First, CRC insertion is done in the CRC Calculate and Insert block **(1)**, next the stream is scrambled in the Scramble block **(2)**, and finally it is byte-swapped in the Swap Bytes Block **(3)**. All three operations can be configured by writing to the *blockControl* register. CRC insertion and scrambling are done as described in CD Yellow Book.

The CRC insertion **(1)** and the Scrambling **(2)** are done on a block-by-block basis. A block is normally 2352 bytes long. A block starts after the so-called *sync Pattern*, “00FFFFFF-FFFFFFFF-FFFFFF00”. To detect the start of a new block, two mechanisms are build into the encoder.

- First long-word of a new block is assumed after finding the sync pattern “00FFFFFF-FFFFFFFF-FFFFFF00”
- First long-word of a new block is assumed exactly 2352 bytes after the first longword of the previous block. This second detection mechanism builds in immunity for corrupted syncs. Even if the sync is corrupted, the block encoder will correctly find the start of the a new block.

17.7.8.2 CD-ROM Encoder Interrupts

- newBlock interrupt—Active when transmission of a new block is started. No direct synchronization with data written to the transmit FIFO.
- noSync interrupt—Set when the sync pattern was not recognized for the current newBlock interrupt.
- ilSync interrupt—Set when the previous block did not have the correct length. (Length different from 2352 bytes).

17.8 DMA Channel Interaction

It is possible to use the DMA to transfer data to/from the FIFO’s in the audio interface module. However, only PDIR2 and PDOR3 registers support DMA transfer, as the others need more than 1 long-word to transfer data to/from the FIFO and cannot be used with DMA operation.

Operation is as follows:

- If PDIR2 is full and DMAConfig(1) is set to ‘0’, DMA1REQ is activated.
- If PDIR2 is full and DMAConfig(0) is set to ‘0’, DMA0REQ is activated.
- If the FIFO connected to PDOR3 is empty, and DMAConfig(1) is set ‘1’, DMA1REQ is activated.
- If the FIFO connected to PDOR3 is empty, and DMAConfig(0) is set to ‘1’, DMA0REQ is activated.

Both DMA1REQ and DMA0REQ can be routed to DMA channel 0 or DMA channel 1.

Address MBAR2 + 0X9F Access: User read/write

	7	6	5	4	3	2	1	0
R							DMA1REQ	DMA0REQ
W								
Reset	–	–	–	–	–	–	0	0

Figure 17-23. DMA Config Register

Table 17-23. DMA Config Register Field Descriptions

Bit Name	Description
DMA1REQ	0 PDIR2 1 PDOR3
DMA0REQ	0 PDIR2 1 PDOR3

17.9 Phase/Frequency Determination and XTRIM Function

The Phase/Frequency determination function can be used to determine when a software sample rate convertor should be enabled and provide the necessary control to steer the sample rate convertor clock (when the incoming sample rate is other than 44.1 kHz). This applies to IIS inputs and the EBU input.

In addition, the Phase/Frequency determination function can also be used to determine when the incoming IEC958 clock does not match the phase of the CRIN clock and use the XTRIM function to trim the CRIN source to match (within a 150ppm range). Typically, when the IEC958 input is being used, the CRIN clock requires trimming to match but this is only when the source is completely external to the application and when any audio output must be synchronous to the input source.

When the source is internal to the application, such as from a CD player controlled by the processor, then the input sample rate does not need to match the output sample rate, they can be asynchronous. When FIFO under-run or over-run occurs, requests can be made to re-read the lost data by the system.

17.9.1 Incoming Source Frequency Measurement

A frequency measurement block exists to allow precise measurement of an incoming sampling frequency. This can be used in conjunction with the XTRIM output (and with the appropriate control s/w) to “lock” the clock being input to CRIN (either external generated clock or crystal) to the recovered SPDIF audio

clock - if so desired. Some external hardware is required for this including a set of varicap diodes.

Upon request Freescale can supply example s/w code that implements the Frequency measurement and controls the XTRIM output to allow CRIN to be locked to the EBU clock input.

The PLL maintaining phase relation between the incoming source signal and internal signal is mainly digital. It is necessary, however, to measure the phase/frequency of the incoming signal in relationship with the CRIN clock in order to be able to steer the sample rate convertor clock. The circuit shown in Figure 17-24 is used for maintaining this phase relationship.

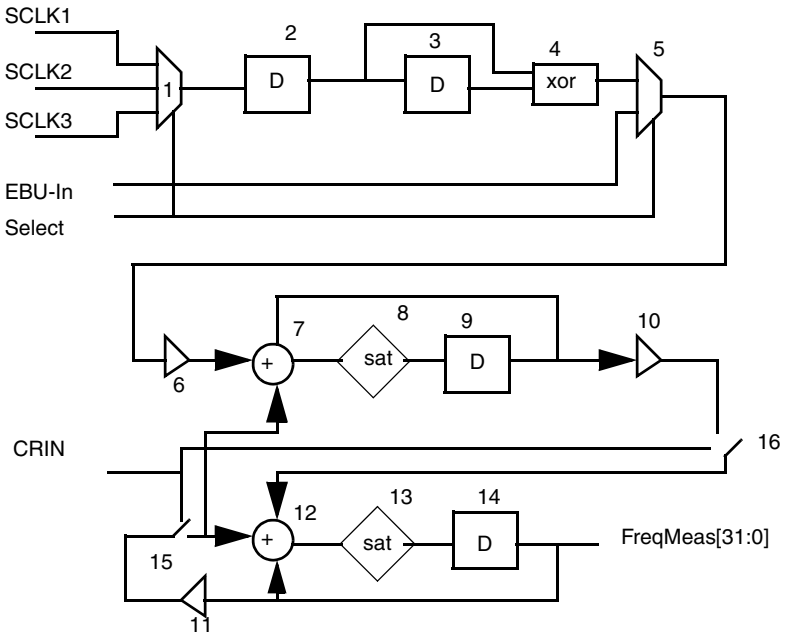


Figure 17-24. Frequency Measurement Circuit

Associated with the Frequency measurement block are two registers. (See Table 17-24). The circuit will measure the frequency of the incoming clock by comparison with the audio CRIN clock (which is typically 16.93 MHz or 11.2896MHz). Multiplexer **(1)** selects the incoming clock source. Registers **(2)**,**(3)**, and xor **(4)** are an edge detector. Multiplexer **(5)** is a by-pass for the IEC958 input. The rest of the circuit is a second-order filter with a bandwidth of approximately 80 Hz. The output FreqMeas(31:0) is an unsigned number, giving the frequency of the selected source as a function of the CRIN clock. The filter is calculated internally in 48 bit precision. The 16 LSBs are not sent out.

The value read from the FreqMeas Register is calculated as follows:

- For measurement of the IIS inputs: FreqMeas = (IIS SCLK Freq × 2)/Faudio × (2 ** 15) × Gain
- For measurement of the EBU input: FreqMeas = (EBU Freq) / Faudio × (2 ** 15) × Gain

Table 17-24. PhaseConfig and Frequency Measure Register Addresses

Address	Name	Width	Description	Access
MBAR2 + 0xA3	PhaseConfig	8	Phase Configuration (gain and source select)	R/W
MBAR2 + 0xA8	FreqMeas	32	Frequency measurement	R

Address MBAR2 + 0XA3 Access: User read/write

	7	6	5	4	3	2	1	0
R			Gain select			Source Select		
W								
Reset	–	–	0	0	0	0	0	0

Figure 17-25. PhaseConfig Register

Table 17-25. PhaseConfig Register Field Descriptions

Bit Name	Description
7–6	Reserved.
5–3 GAIN SELECT	000 6 * 2 ** 15 001 4 * 2 ** 15 010 3 * 2 ** 15 011 4 * 2 ** 14 100 3 * 2 ** 14 101 4 * 2 ** 13 110 3 * 2 ** 13
2–0 SOURCE SELECT	000 SCLK1 001 SCLK2 010 SCLK3 011 Reserved 100 EBUIN Others Reserved, undefined

17.9.1.1 Filtering for the Discrete Time Oscillator

The frequency measurement circuit first detects the edges of the incoming clock. This pulse signal is then passed through an 80 Hz band-width low-pass filter. The signal that comes after the low-pass filter is low-noise, and is suitable for precision frequency measurement. (Expected noise level: order-of magnitude -100 dB). Before it can be used as a frequency increment for the Discrete Time oscillator, it must undergo additional filtering to push back the noise level on the phase.

17.9.2 XTRIM Option - Locking Xtal Clock to Incoming Signal

The XTRIM output allows use of varicap-controlled crystal. (See Figure 17-26). To do this, the XTRIM must output a PWM/PDM modulated phase-error signal. One 16-bit config register is associated with this functionality.

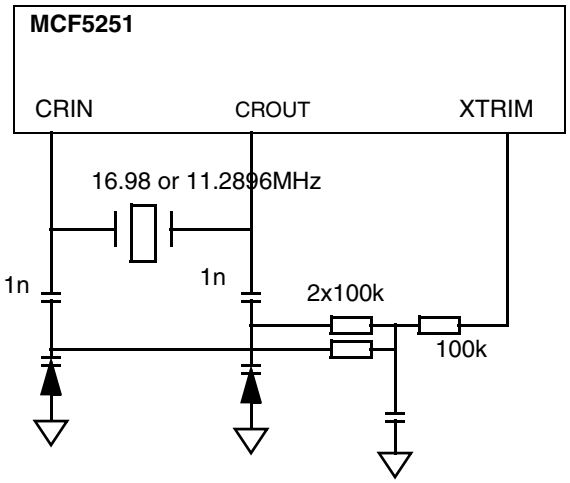


Figure 17-26. XTRIM External Circuit

Table 17-26. XTRIM Register Address and Description

Address	Name	Width	Description	Reset Value	Access
MBAR2 + 0xA6	XTRIM	16	XTRIM output value	0x8000	RW

The duty cycle output on XTRIM is proportional to the value written to register XTIM. 0x0000 corresponds with duty cycle 0, 0xFFFF corresponds with 100%. 0x8000 corresponds with 50%.

17.9.3 XTRIM Internal Logic

For XTRIM, the internal circuit of the PDM modulator is used as shown in Figure 17-27. It is a first-order pulse density modulator, working from the system clock divided by 16.

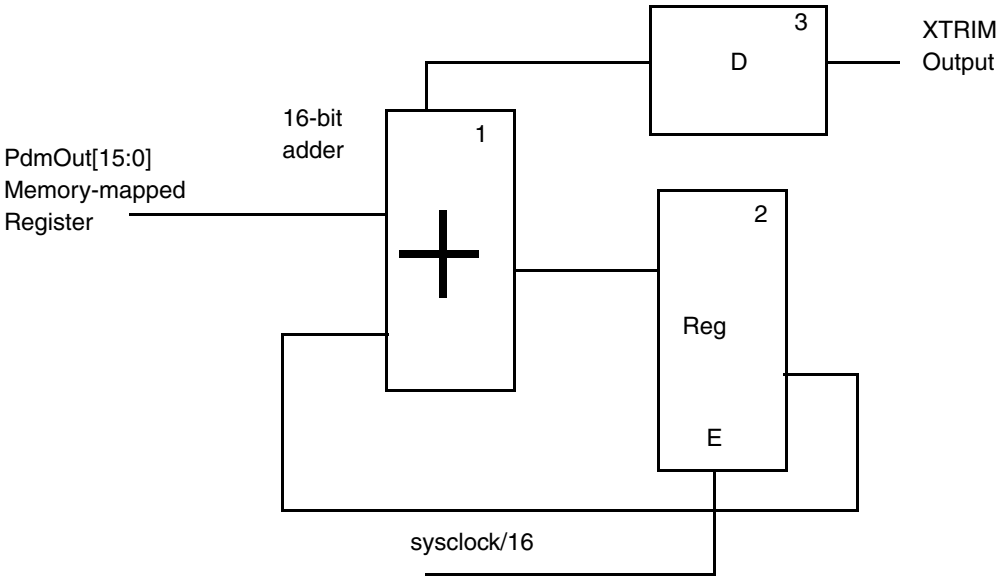


Figure 17-27. PDM Modulator Used on XTRIM Output

Chapter 18
I²C Modules

This chapter provides the system configuration and protocol of the Inter IC Communications (I²C) modules of the MCF5251, the memory map and register descriptions, and a programming example.

18.1 I²C Interface Features

- Compatibility with I²C Bus standard
- Multimaster operation
- Software-programmable for one of 64 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

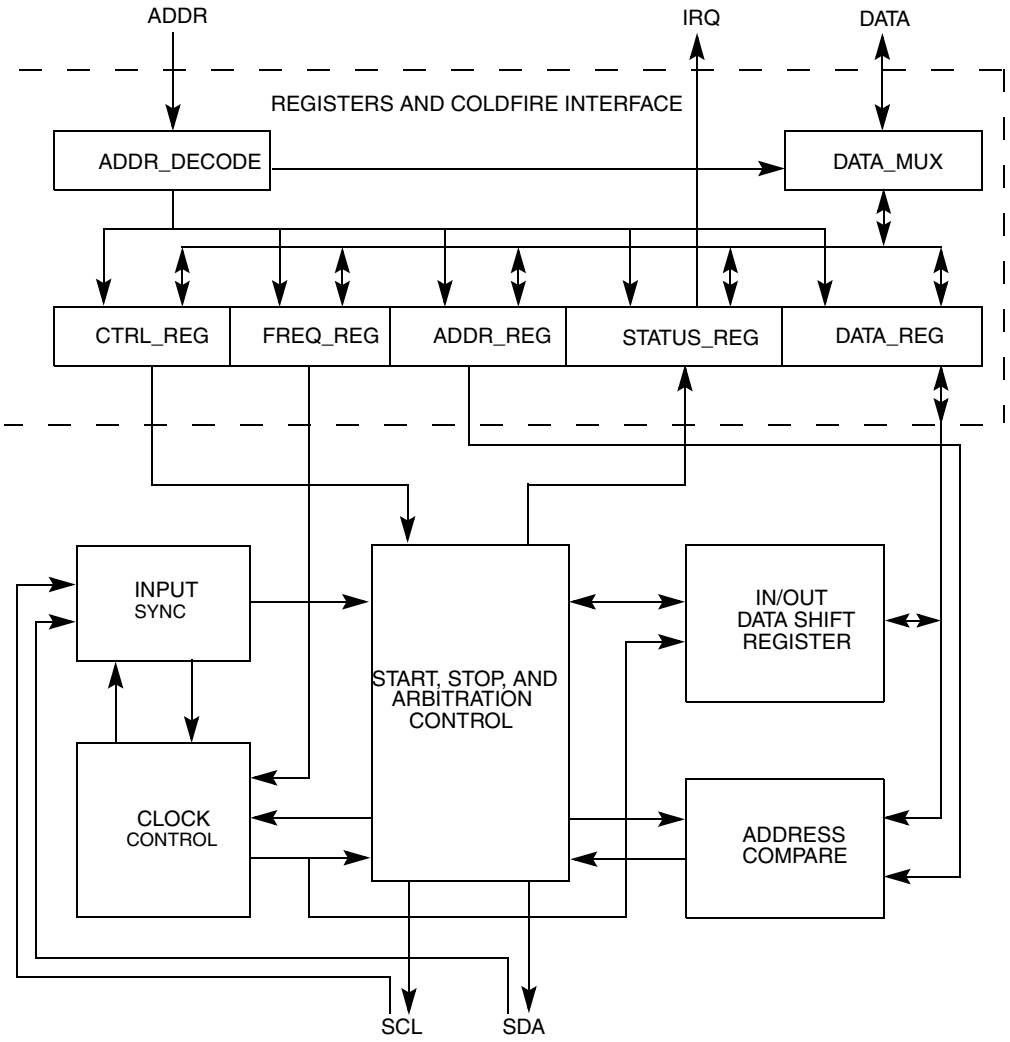


Figure 18-1. I²C Module Block Diagram

18.2 I²C Overview

The MCF5251 provides dual I²C interface capability. The I²C interface described in this chapter is fully compatible with the I²C Bus Standard.

The I²C is a two-wire, bidirectional serial bus that provides a simple and efficient method of data exchange between devices. This two-wire bus minimizes the interconnection between devices.

The I²C bus is suitable for applications requiring occasional communications over a short distance between many devices. The flexible I²C allows additional devices to be connected to the bus for expansion and system development.

The interface operates up to 100 kbps with maximum bus loading and timing. Operation can be extended to operate at the high speed I²C specification (400 kbps) but this requires careful thought and design. To adhere to the I²C bus timing specification load capacitance will have to be carefully controlled, to this end the number of devices that can be attached to a high speed I²C bus interface will be limited. Also the value

of the I²C pull-up resistors will need to be kept at a relatively low impedance, which may cause noise issues in some systems.

The I²C system is a true multimaster bus including collision detection and arbitration that prevents data corruption if two or more masters attempt to control the bus simultaneously. This feature allows for complex applications with multiprocessor control. It can also be used for rapid testing and alignment of end products using external connections to an assembly line computer.

18.3 I²C System Configuration

I²C module uses a serial data line (SDA) and a serial clock line (SCL) for data transfer. 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 pullup resistors.

The default state of I²C is as a slave receiver out of reset. Thus, when not programmed to be a master or responding to a slave transmit address, the I²C module should always return to the default state of slave receiver.

NOTE

This I²C module is designed to be compatible with the I²C bus protocol from Philips. For further information on I²C system configuration, protocol, and restrictions please refer to the Philips I²C Standard.

18.4 I²C Protocol

A standard communication is composed of four parts:

1. START signal
2. Slave address transmission
3. Data transfer
4. STOP signal

They are described briefly in the following sections and shown in Figure 18-2.

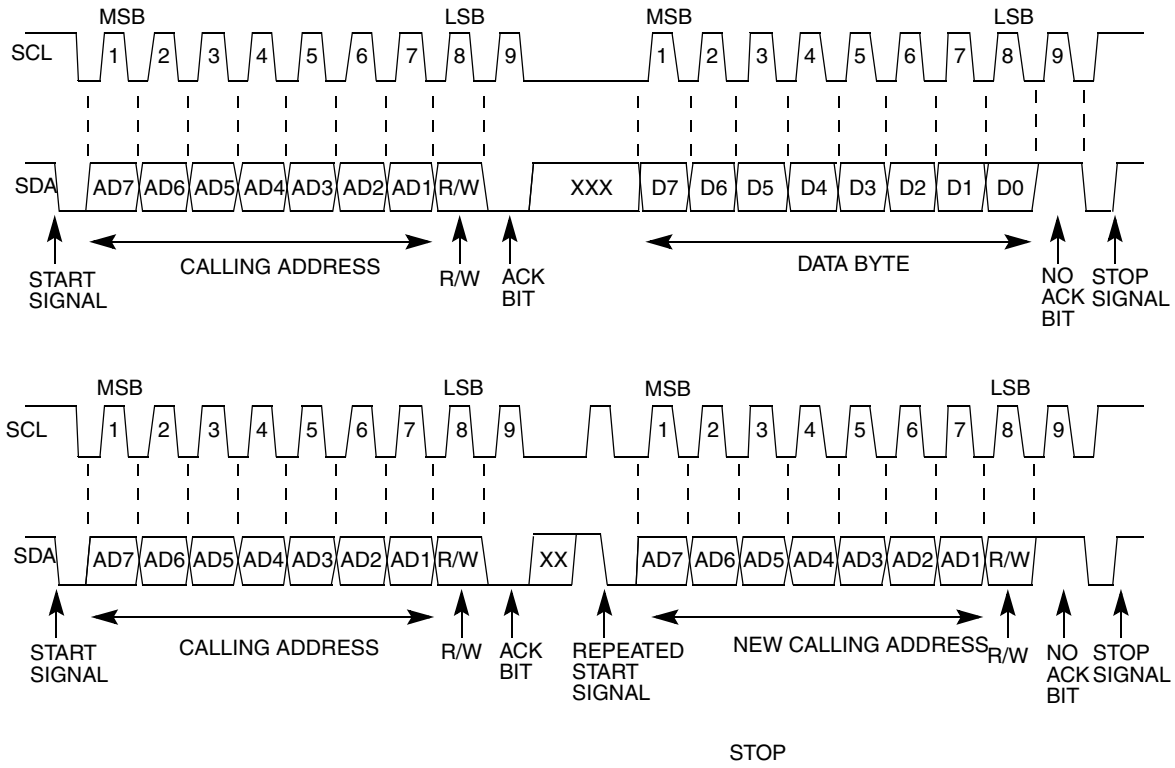


Figure 18-2. I²C Standard Communication Protocol

18.4.1 START Signal

When the bus is free, for example, no master device is engaging the bus (both SCL and SDA lines are at logic high), a master can initiate communication by sending a START signal. As shown in Figure 18-2, a START signal is defined as a high-to-low transition of SDA while SCL is high. This signal denotes the beginning of a new data transfer (each data transfer can contain several bytes of data) and awakens all slaves.

18.4.2 Slave Address Transmission

The first byte of data transferred by the master immediately after the START signal is the slave address. This is a seven-bit calling address followed by a R/W bit. The R/W bit tells the slave data transfer direction. No two slaves in the system can have the same address. In addition, if the I²C is master, it must not transmit an address that is equal to its slave address. The I²C cannot be master and slave at the same time.

Only the slave with an address that matches the one transmitted by the master will respond. It returns an acknowledge bit by pulling the SDA low at the 9th clock (see Figure 18-2).

18.4.3 Data Transfer

Once successful slave addressing is achieved, the data transfer can proceed on a byte-by-byte basis in the direction specified by the R/W bit sent by the calling master.

Each data byte is 8 bits long. Data can be changed only while SCL is low and must be held stable while SCL is high, as shown in Figure 18-2 There is one clock pulse on SCL for each data bit with the MSB being transferred first. Each byte of data must be followed by an acknowledge bit, which is signalled from the receiving device by pulling the SDA low at the ninth clock. One complete data byte transfer needs nine clock pulses.

If the slave receiver does not acknowledge the master, the SDA line must be left high by the slave. The master can then generate a stop signal to abort the data transfer or a start signal (repeated start) to start a new calling sequence.

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 the SDA line for the master to generate a STOP or START signal.

18.4.4 Repeated START Signal

As shown in Figure 18-2, a repeated START signal is a START signal generated without first generating a STOP signal to terminate the communication. The master uses this method to communicate with another slave or with the same slave in a different mode (transmit/receive mode) without releasing the bus.

18.4.5 STOP Signal

The master can terminate the communication by generating a STOP signal to free the bus. However, the master can generate a START signal followed by a calling command without generating a STOP signal first. This is called repeated START. A STOP signal is defined as a low-to-high transition of SDA while SCL is at logical 1 (see Figure 18-2).

NOTE

A master can generate a STOP even if the slave has made an acknowledgment at which point the slave must release the bus.

18.4.6 Arbitration Procedure

I²C is a true multimaster bus that allows connection to more than one master. If two or more masters try to simultaneously control the bus, a clock synchronization procedure determines the bus clock, for which the low period is equal to the longest clock low period and the high is equal to the shortest one among the devices. 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 SDA output. In this case, the transition from master to slave mode does not generate a STOP condition. Meanwhile, hardware sets MBSR[IAL] to indicate loss of arbitration.

18.4.7 Clock Synchronization

Because wire-AND logic is performed on SCL line, a high-to-low transition on SCL line affects all the devices connected on the bus. The devices start counting their low period when the master drives the SCL line low. Once a device clock has gone low, it holds the SCL line low until the clock high state is reached. However, the change of low to high in the MCF5251 clock may not change the state of the SCL line if

another device clock is still within its low period. Therefore, 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 18-3). When all devices concerned have counted off their low period, the SCL line is released and pulled high. At this point, there is no difference between the device clocks and the state of the SCL line and all the devices start counting their high periods. The first device to complete its high period pulls the SCL line low again.

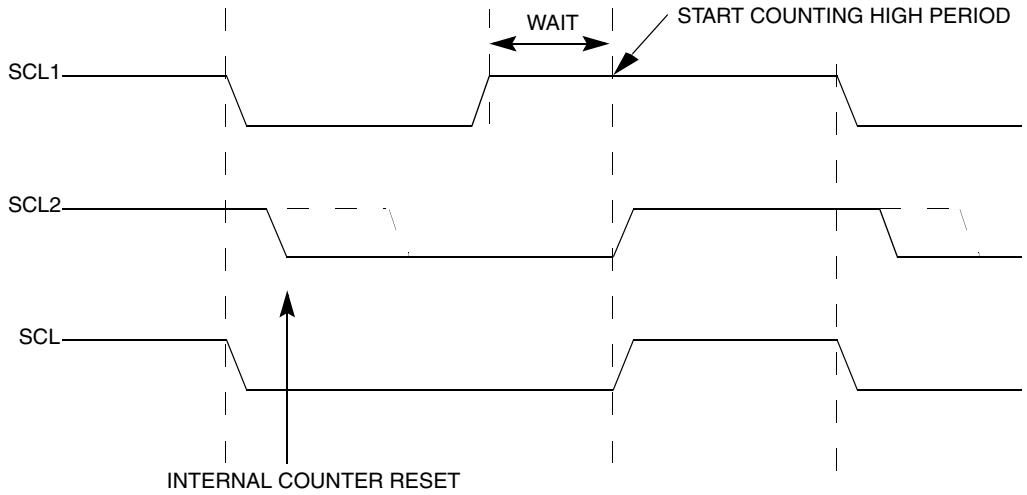


Figure 18-3. Synchronized Clock SCL

18.4.8 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfer. Slave devices can hold the SCL line low after completion of one byte transfer (9 clocks). In such cases, it halts the bus clock and forces the master clock into wait states until the slave releases the SCL line.

18.4.9 Clock Stretching

Slaves can use the clock synchronization mechanism to slow down the transfer bit rate. After the master has driven SCL low, the slave can drive SCL low for the required period and then release it. If the slave SCL low period is greater than the master SCL low period, the resulting SCL bus signal low period is stretched.

18.5 I²C Memory Map and Register Descriptions

Internal configuration of the five registers used in the I²C interface are detailed in the following sections. Table 18-1 shows the register summary of the I²C interface.

Table 18-1. I²C Interfaces Register Summary

Address	I²C Module Registers
MBAR+$280	I²C Address Register (MADR)
MBAR+$284	I²C Frequency Divider Register (MFDR)

Table 18-1. I²C Interfaces Register Summary (continued)

Address	I²C Module Registers
MBAR+$288	I²C Control Register (MBCR)
MBAR+$28C	I²C Status Register (MBSR)
MBAR+$290	I²C Data I/O Register (MBDR)
MBAR2 + $440	MBAR2 I²C Address Register (MADR2)
MBAR2 + $444	MBAR2 I²C Frequency Divider Register (MFDR2)
MBAR2 + $448	MBAR2 I²C Control Register (MBCR2)
MBAR2 + $44C	MBAR2 I²C Status Register (MBSR2)
MBAR2 + $450	MBAR2 I²C Data I/O Register (MBDR2)

NOTE

External masters cannot access the MCF5251 on-chip memories or MBAR, but can access any I²C module register.

18.5.1 I²C Address Registers (MADR)

This register contains the address that the I²C will respond to when addressed as a slave.

NOTE

It is not the address sent on the bus during the address transfer.

Address MBAR+$280 (MADR)
MBAR2+$440 (MADR2)

Access: Supervisor or User read/write

	7	6	5	4	3	2	1	0
R	ADR7	ADR6	ADR5	ADR4	ADR3	ADR2	ADR1	
W								
Reset	0	0	0	0	0	0	0	0

Figure 18-4. MADR Register

Table 18-2. MADR Register Field Descriptions

Field	Description
7–1 ADR	Slave Address. Bit 1 to bit 7 contains the specific slave address to be used by the I²C module.
0	Reserved.

18.5.2 I²C Frequency Divider Registers (MFDR)

The MFDR provides a programmable prescalar to configure the clock for bit rate selection.

Address MBAR+ $284 (MFDR)
MBAR2+ $444 (MFDR2)

Access: Supervisor or User read/write

	7	6	5	4	3	2	1	0
R			IC5	IC4	IC3	IC2	IC1	IC0
W								
Reset	0	0	0	0	0	0	0	0

Figure 18-5. MFDR Register

Table 18-3. MFDR Register[1] Field Descriptions

Field	Description
7–6	Reserved
5–0 IC	I²C Clock Rate. This field is used to prescale the clock for bit rate selection. Due to the potential slow rise and fall times of the SCL and SDA signals, bus signals are sampled at the prescaler frequency. The serial bit clock frequency is equal to the system clock divided by the divider shown in Table 18-4

[1] The MFDR frequency value can be changed at any point in a program.

Table 18-4. I²C Prescaler Values

MBC5-0 (hex)	Divider (dec)		MBC5-0 (hex)	Divider (dec)
00	28		20	20
01	30		21	22
02	34		22	24
03	40		23	26
04	44		24	28
05	48		25	32
06	56		26	36
07	68		27	40
08	80		28	48
09	88		29	56
0A	104		2A	64
0B	128		2B	72
0C	144		2C	80
0D	160		2D	96
0E	192		2E	112
0F	240		2F	128
10	288		30	160
11	320		31	192

Table 18-4. I²C Prescaler Values (continued)

MBC5-0 (hex)	Divider (dec)		MBC5-0 (hex)	Divider (dec)
12	384		32	224
13	480		33	256
14	576		34	320
15	640		35	384
16	768		36	448
17	960		37	512
18	1152		38	640
19	1280		39	768
1A	1536		3A	896
1B	1920		3B	1024
1C	2304		3C	1280
1D	2560		3D	1536
1E	3072		3E	1792
1F	3840		3F	2048

18.5.3 I²C Control Registers (MBCR)

The MBCR enables the I²C module and the associated I²C interrupts. It also contains the bits that govern operation as Master or Slave.

Address MBAR+ $288 (MBCR)
MBAR2+ $448 (MBCR2)

Access: Supervisor or User read/write

	7	6	5	4	3	2	1	0
R	IEN	IIEN	MSTA	MTX	TXAK	RSTA		
W								
Reset	0	0	0	0	0	0	0	0

Figure 18-6. MBCR Register

Table 18-5. MBCR Register Field Descriptions

Field	Description
7 IEN	The I²C Enable bit controls the software reset of the entire I²C module. 1 The I²C module is enabled. This bit must be set before any other MBCR bits have any effect. 0 The module is disabled, but registers can still be accessed. If the I²C module is enabled in the middle of a byte transfer, the interface behaves as follows: The slave mode ignores the current transfer on the bus and starts operating whenever a subsequent start condition is detected. Master mode will not be aware that the bus is busy; therefore, if a start cycle is initiated, the current bus cycle can become corrupt. This ultimately results in either the current bus master or the I²C module losing arbitration, after which bus operation returns to normal.
6 IIEN	I²C Interrupt Enable 1 Interrupts from the I²C module are enabled. An I²C interrupt occurs provided the IIF bit in the status register is also set. 0 Interrupts from the I²C module are disabled. This does not clear any currently pending interrupt condition.
5 MSTA	At reset, the Master/Slave Mode Select Bit is cleared. When this bit is changed from 0 to 1, a START signal is generated on the bus, and the master mode is selected. When this bit is changed from 1 to 0, a STOP signal is generated and the operation mode changes from master to slave. MSTA is cleared without generating a STOP signal when the master loses arbitration. 1 Master Mode 0 Slave Mode
4 MTX	The Transmit/Receive Mode Select Bit selects the direction of master and slave transfers. When addressed as a slave this bit should be set by software according to the SRW bit in the status register. In master mode, this bit should be set according to the type of transfer required. Therefore, for address cycles, this bit will always be high. 1 Transmit 0 Receive
3 TXAK	The Transmit Acknowledge Enable bit specifies the value driven onto SDA during acknowledge cycles for both master and slave receivers. Writing this bit only applies when the I²C bus is a receiver, not a transmitter. 1 No acknowledge signal response is sent (i.e., acknowledge bit = 1) 0 An acknowledge signal will be sent out to the bus at the 9th clock bit after receiving one byte data
2 RSTA	Writing a 1 to the Repeat Start bit will generate a repeated START condition on the bus, provided it is the current bus master. This bit will always be read as a low. Attempting a repeated start at the wrong time, if the bus is owned by another master, will result in loss of arbitration. 1 Generate repeat start cycle 0 No repeat start
1–0	Reserved.

18.5.4 I²C Status Registers (MBSR)

This status register is read-only with the exception of bit 1 (IIF) and bit 4 (IAL), which can be cleared by software. All bits are cleared on reset except bit 7 (ICF) and bit 0 (RXAK), which are set (=1) at reset.

Address MBAR+ $28C (MBSR)
MBAR2+ $44C (MBSR2)

Access: Supervisor or User read/write

	7	6	5	4	3	2	1	0
R	ICF	IAAS	IBB	IAL		SRW	IIF	RXAK
W								
Reset	1	0	0	0	0	0	0	1

Figure 18-7. MBSR Register

Table 18-6. MBSR Register Field Descriptions

Field	Description
7 ICF	While one byte of data is being transferred, the Data Transferring Bit bit is cleared. It is set by the falling edge of the 9th clock of a byte transfer. 1 Transfer complete 0 Transfer in progress
6 IAAS	When its own specific address (I²C Address Register) is matched with the calling address, the Addressed as a Slave Bit is set. The CPU is interrupted provided the IIEN is set. Next, the CPU must check the SRW bit and set its TX/RX mode accordingly. Writing to the I²C Control Register clears this bit. 1 Addressed as a slave 0 Not addressed
5 IBB	The Bus Busy Bit indicates the status of the bus. When a START signal is detected, the IBB is set. If a STOP signal is detected, it is cleared. 1 Bus is busy 0 Bus is idle
4 IAL	Hardware sets the Arbitration Lost bit (IAL) when the arbitration procedure is lost. Arbitration is lost in the following circumstances: • SDA sampled as low when the master drives a high during an address or data-transmit cycle. • SDA sampled as a low when the master drives a 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. This bit must be cleared by software by writing a zero to it.
3	Reserved
2 SRW	When IAAS is set, the Slave Read/Write bit indicates the value of the R/W command bit of the calling address sent from the master. This bit is valid only when: • A complete transfer has occurred and no other transfers have been initiated. • I²C is a slave and has an address match. Checking this bit, the CPU can select slave transmit/receive mode according to the command of the master. 1 Slave transmit, master reading from slave 0 Slave receive, master writing to slave

Table 18-6. MBSR Register Field Descriptions

Field	Description
1 IIF	The I²C Interrupt (IIF) bit is set when an interrupt is pending, which will cause a processor interrupt request (provided IIEN is set). IIF is set when one of the following events occurs: • Complete one byte transfer (set at the falling edge of the 9th clock) • Receive a calling address that matches its own specific address in slave-receive mode • Arbitration lost This bit must be cleared by software by writing a zero to it in the interrupt routine.
0 RXAK	The value of SDA during the acknowledge bit of a bus cycle. If the received acknowledge bit (RXAK) is low, it indicates an acknowledge signal has been received after the completion of 8 bits data transmission on the bus. If RXAK is high, it means no acknowledge signal has been detected at the 9th clock. 1 No acknowledge received 0 Acknowledge received

18.5.5 I²C Data I/O Registers (MBDR)

When an address and R/W bit is written to the MBDR and the I²C is the master, a transmission will start. When data is written to the MBDR, a data transfer is initiated. The most significant bit is sent first in both cases. In the master-receive mode, reading the MBDR register allows the read to occur but also initiates next byte data receiving. In slave mode, the same function is available after it is addressed.

Address MBAR+$290 (MBDR)
MBAR2+$450 (MBDR2)

Access: Supervisor or User read/write

	7	6	5	4	3	2	1	0
R W	D7	D6	D5	D4	D3	D2	D1	D0
Reset	0	0	0	0	0	0	0	0

Figure 18-8. MBDR Register

18.6 I²C Programming Examples

18.6.1 Initialization Sequence

A reset places the I²C Control Register into default status. Before the interface can transfer serial data, users must perform an initialization procedure as follows:

1. Update the Frequency Divider Register (MFDR) and select the required division ratio to obtain SCL frequency from the system bus clock.
2. Update the I²C Address Register (MADR) to define its slave address.
3. Set the IEN bit of the I²C Control Register (MBCR) to enable the I²C bus interface system.
4. Modify the MBCR to select master/slave mode, transmit/receive mode, and interrupt-enable or not.

NOTE

During the initialization of the I²C bus module, the user should check the IBB bit of the MBSR register. If the IBB bit is set when the I²C module is enabled, then the following code sequence should be executed before proceeding with the normal initialization code. This issues a STOP command to the slave device, which places it into the idle state as if it were recently power cycled.

```
MBCR = $0
MBCR = $A0
dummy read of MBDR
MBSR = $0
MBCR = $0
```

18.6.2 Generation of START

After completion of the initialization procedure, users can transmit serial data by selecting the “master transmitter'' mode. If the MCF5251 is connected to a multimaster bus system, users must test the state of the I²C Busy Bit (IBB) to check whether the serial bus is free.

If the bus is free (IBB=0), the start condition 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 is set to indicate the direction of transfer required.

The bus free time (i.e., the time between a STOP condition and the following START condition) is built into the hardware that generates the START cycle. Depending on the relative frequencies of the system clock and the SCL period, users may have to wait until the I²C is busy after writing the calling address to the MBDR before proceeding with the following instructions.

An example of a program that generates the START signal and transmits the first byte of data (slave address) is shown as follows:

```
CHFLAG MOVE.B  MBSR,-(A7);Check the MBB bit of the MBSR
BTST.B #5, (A7)+
BNE.S CHFLAG;If it is set, wait until it is clear

TXSTART MOVE.B MBCR,-(A7);Set transmit mode
BSET.B #4,(A7)
MOVE.B (A7)+, MBCR
MOVE.B MBCR, -(A7);Set master mode
BSET.B #5, (A7);Generate START condition

MOVE.B (A7)+, MBCR;
MOVE.B CALLING,-(A7);Transmit the calling address, D0=R/W
MOVE.B (A7)+, MBDR;

IFREE MOVE.B MBSR,-(A7);Check the IBB bit of the MBSR.
;If it is clear, wait until it is set.
BTST.B #5, (A7)+;
BEQ.S IBFREE;
```

18.6.3 Post-Transfer Software Response

Transmission or reception of a byte will set the data transferring bit (ICF) to 1, which indicates one byte communication is finished. The interrupt bit (IIF) is also set. An interrupt will be generated if the interrupt function is enabled during initialization by setting the IIEN bit. Software must clear the IIF bit in the interrupt routine first. The ICF bit will be cleared by reading from the I²C Data I/O Register (MBDR) in receive mode or writing to MBDR in transmit mode.

Software can service the I²C I/O in the main program by monitoring the IIF bit if the interrupt function is disabled. Polling should monitor the IIF bit rather than the ICF bit because that operation is different when arbitration is lost.

When an interrupt occurs at the end of the address cycle, the master will always be in transmit mode. For example, the address is transmitted. If master receive mode is required, indicated by MBDR[R/W], then the MTX bit should be toggled.

During slave-mode address cycles (IAAS=1), the SRW bit in the status register is read to determine the direction of the subsequent transfer and the MTX bit is programmed accordingly. For slave-mode data cycles (IAAS=0), the SRW bit is not valid. The MTX bit in the control register should be read to determine the direction of the current transfer.

The following is an example of a software response by a “master transmitter”' in the interrupt routine (see Figure 18-9).

```
MBSR     LEA.L    MBSR,-(A7)         ;Load effective address
         BCLR.B   #1,(A7)+           ;Clear the IIF flag
         MOVE.B   MBCR,-(A7)         ;Push the address on stack,
         BTST.B   #5,(A7)+           ;check the MSTA flag
         BEQ.S    SLAVE              ;Branch if slave mode
         MOVE.B   MBCR,-(A7)         ;Push the address on stack
         BTST.B   #4,(A7)+           ;check the mode flag
         BEQ.S    RECEIVE            ;Branch if in receive mode
         MOVE.B   MBSR,-(A7)         ;Push the address on stack,
         BTST.B   #0,(A7)+           ;check ACK from receiver
         BNE.B
END                        ;If no ACK, end of transmission
TRANSMIT MOVE.B   DATABUF,-(A7)      ;Stack data byte
         MOVE.B   (A7)+, MBDR)       ;Transmit next byte of data
```

18.6.4 Generation of STOP

A data transfer ends with a STOP signal generated by the “master”' device. A master transmitter can generate a STOP signal after all the data has been transmitted. The following code is an example showing how a master transmitter generates a stop condition.

```
MASTX    MOVE.B   MBSR, -(A7)        ; If no ACK, branch to end
         BTST.B   #0,(A7)+
         BNE.B    END
         MOVE.B   TXCNT,D0           ;Get value from the transmitting counter
         BEQ.S    END                ;If no more data, branch to end
         MOVE.B   DATABUF,-(A7)      ;Transmit next byte of data
         MOVE.B   (A7)+,MBDR
         MOVE.B   TXCNT,D0           ;Decrease the TXCNT
         SUBQ.L   #1,D0
```

```
          MOVE.B   D0,TXCNT
          BRA.S    EMASTX            ;Exit
END       LEA.L    MBCR,-(A7)        ;Generate a STOP condition
          BCLR.B   #5,(A7)+
EMASTX    RTE;                       Return from interrupt
```

If a master receiver wants to terminate a data transfer, it must inform the slave transmitter by not acknowledging the last byte of data, which can be done by setting the transmit acknowledge bit (TXAK) before reading the next-to-last byte of data. Before reading the last byte of data, a STOP signal must first be generated. The following code is an example showing how a master receiver generates a STOP signal.

```
MASR      MOVE.B   RXCNT,D0          ;Decrease RXCNT
          SUBQ.L   #1,D0
          MOVE.B   D0,RXCNT
          BEQ.S    ENMASR            ;Last byte to be read
          MOVE.B   RXCNT,D1          ;Check second-to-last byte to be read
          EXTB.L   D1
          SUBI.L   #1,D1             ;
          BNE.S    NXMAR             ; Not last one or second last
LAMAR     BSET.B   #3,MBCR           ;Disable ACK
          BRA      NXMAR
ENMASR    BCLR.B   #5,MBCR           ; Last one, generate 'STOP'signal
NXMAR     MOVE.B   MBDR,RXBUF        ; Read data and store RTE
```

18.6.5 Generation of Repeated START

At the end of data transfer, if the master still wants to communicate on the bus, it can generate another START signal followed by another slave address without first generating a STOP signal. A program example follows.

```
RESTART   MOVE.B   MBCR,-(A7)        ; Another START (RESTART)
          BSET.B   #2, (A7)
          MOVE.B   (A7)+, MBCR
          MOVE.B   CALLING,-(A7)     ;Transmit the calling address, D0=R/W-
          MOVE.B   CALLING,-(A7)     ;
          MOVE.B   (A7)+, MBDR
```

18.6.6 Slave Mode

In the slave interrupt service routine, the module that is addressed as slave bit (IAAS), should be tested to check if a calling of its own address was received. If IAAS is set, software should set the transmit/receive mode select bit (MTX bit of MBCR) according to the R/W command bit (SRW). Writing to the MBCR clears the 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 will have IAAS cleared. A data transfer can now be initiated by writing information to MBDR, for slave transmits, or read from MBDR, in slave-receive mode. A dummy read of the MBDR in slave/receive mode will release SCL, allowing the master to transmit data.

In the slave transmitter routine, the received acknowledge bit (RXAK) must be tested before transmitting the next byte of data. Setting RXAK means an “end-of-data'' signal from the master receiver, after which it must be switched from transmitter mode to receiver mode by software. A read from MBDR then releases the SCL line so that the master can generate a STOP signal.

18.6.7 Arbitration Lost

If several devices try to engage the bus at the same time, only one becomes master and the others lose arbitration. The devices that lost arbitration are immediately switched to slave receive mode by the hardware. Their data output to the SDA line is stopped, but SCL is still generated until the end of the byte during which arbitration was lost. An interrupt occurs at the falling edge of the ninth clock of this transfer with IAL=1 and MSTA=0. If one master tries to transmit or do a START while the bus is being engaged by another master, the hardware does the following:

1. Inhibits the transmission
2. Switches the MSTA bit from 1 to 0 without generating STOP condition
3. Generates an interrupt to CPU
4. Sets the IAL to indicate the failed attempt to engage the bus

When considering these cases, the slave service routine should test the IAL first and the software should clear the IAL bit if it is set.

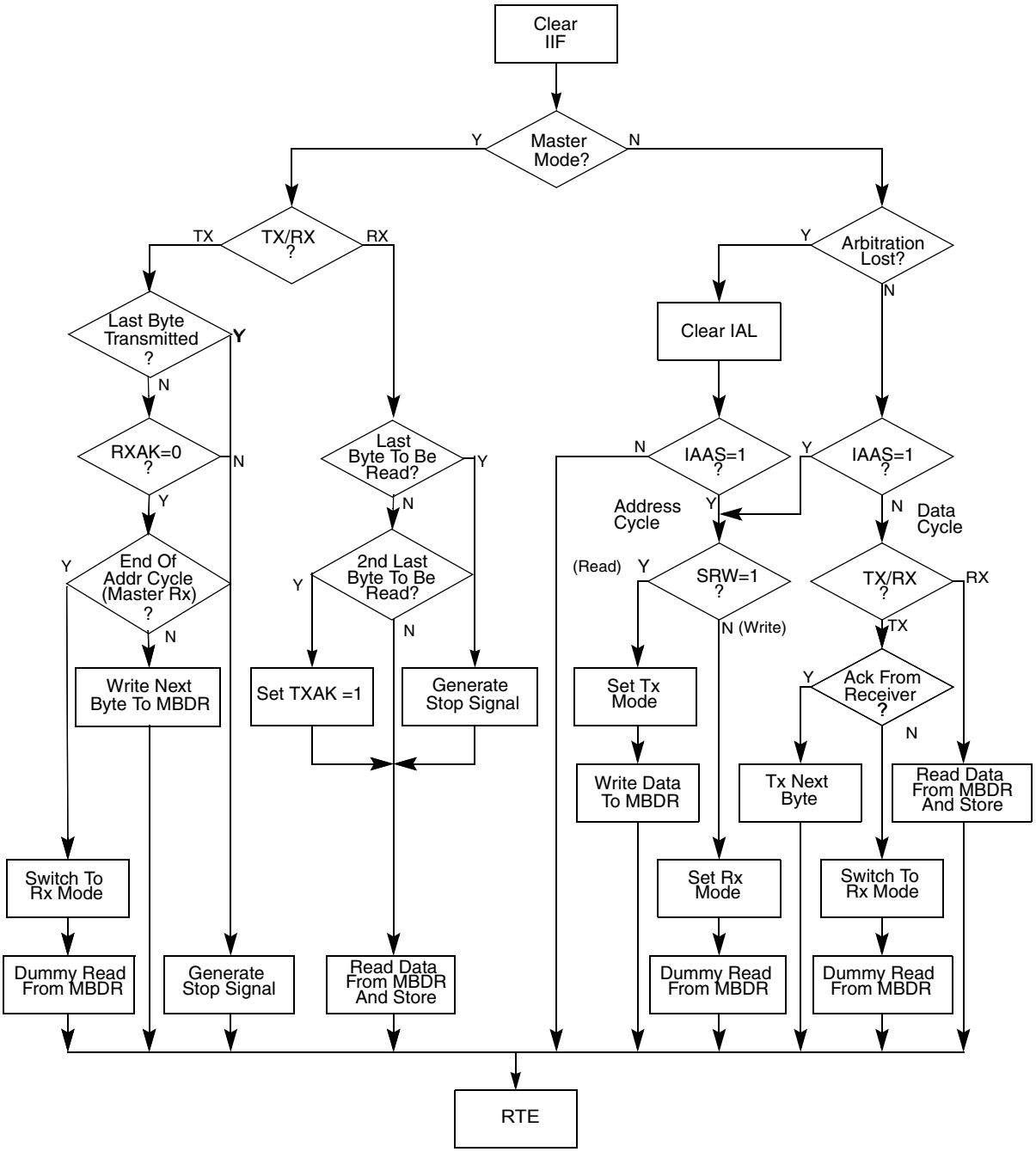


Figure 18-9. Flow-Chart of Typical I²C Interrupt Routine

Chapter 19
Boot ROM

This chapter describes the BootROM operation, the boot modes, and creation of record files.

19.1 Overview

The boot ROM on the MCF5251 serves to boot the CPU in designs which do not have external Flash memory or ROM. Typically these systems use a separate MCU for control, and/or the MCF5251 is used as a stand-alone decoder.

The boot loader resides in 8 K byte on-chip ROM and is enabled by means of a pull-down resistor connected to address pin A23.

With the exception of the UART modes, the boot ROM assumes that the maximum input clock frequency will be 33.8688 MHz.

Therefore, in I^2C master or SPI Master modes the generated serial clock will scale with the input clock frequency.

For the UART boot modes there are three different settings available to allow operation with the following crystal or external clock frequencies: 5, 5.6448, 8.4672, 10, 11.2896, 16.9344, 20 and 33.8688 MHz.

19.1.1 Boot Modes

The MCF5251 can be booted in one of three modes:

- External ROM
- Internal ROM Master mode
- Internal ROM Slave mode

Boot from external ROM is selected by pulling A23 high during system Power-on Reset. In this mode, the internal boot ROM is not available in the memory map, and the boot process is under control of the user program in external memory connected to CS0.

By pulling A23 low during Power-on Reset, CS0 is assigned to the internal boot ROM, where code execution will begin after RESET.

Pin $\overline{CS0}/\overline{CS4}$ is assigned the function of CS4.

The internal boot ROM code supports master and slave modes.

In Master mode operation, the MCF5251 will control the communication with the boot device. It will attempt to load code from a slave device to RAM (either internal or external) and execute this code when it receives an execute command. In this mode, no error recovery is performed by the boot loader.

This mode supports booting from the following:

- I^2C connected slave device
- SPI connected slave device
- Hard Disk Drive (IDE)

In slave mode, the MCF5251 will wait for communication from a controlling MCU and store the code it receives in RAM. It executes this code after receiving an execute command. The controlling MCU must control the data transfer and handle error recovery if required.

This mode supports booting from the following:

- I^2C connected master device
- UART connected master

19.2 Boot ROM Operation

19.2.1 Initialization

19.2.1.1 Boot ROM Memory map

```
bootrom:origin = 0x00000000, length = 0x00002000
mbar:origin = 0xb0000000, length = 0x10000000
mbar2:origin = 0xc0000000, length = 0x40000000
sram1:origin = 0x10000000, length = 0x00010000
sram0:origin = 0x10010000, length = 0x00010000
```

19.2.1.2 Internal SRAM usage

The boot ROM data resides in the upper range of the internal SRAM.

The SRAM allocation is as follows:

```
0x1001FC90 - 0x1001FF13Bootloader uninitialized data (HDD) (644 bytes)
0x1001FF14 - 0x1001FF3FBootloader uninitialized data (Serial) (44 bytes)
0x1001FF40 - 0x1001FFFFBootloader Stack (192 bytes)
```

NOTE

The HDD data space is available for use when booting from a serial device. If the user routine exceeds the assigned stack allocation, then it would be more appropriate for the user to define their own stack space, especially if the user routine returns to the bootloader.

```
Initialization sequence
_Boot:
        move.w   #0x2700,SR
        move.l   #VECTOR_TABLE,d0
        movec    d0,VBR

;       /* Invalidate the cache and disable it */
        move.l   #0x01000000,d0
        movec    d0,cacr
```

```
;       /* Disable ACRs */
        moveq.l  #0,d0
        movec    d0,ACR0
        movec    d0,ACR1

;       /* Initialize SRAMBAR */
        move.l   #SRAM0base+1,d0;  /* locate SRAM0, validate it! */
        movec    d0,SRAMBAR
        move.l   #SRAM1base+1,d0;  /* locate SRAM1, validate it! */
        movec    d0,SRAMBAR1

        move.l   #___MBAR+1,d0;    /* locate MBAR and validate */
        movec    d0,MBAR;
        move.l   #___MBAR2+1,d0;   /* locate MBAR2 and validate*/
        movec    d0,MBAR2;

;       /* Initialize CS0 */
        move.l   #ROMbase,d0;
        move.l   d0,CSAR0;         /* locate ROM */
        move.l   #0xFFFF0001,d0;   /* block size 64 KB, validate */
        move.l   d0,CSMR0;
        move.l   #0x0580,d0;       /* port size 16 bit, AA, 1WS */
        move.l   d0,CSCR0;

        move.l   #___SP_INIT,sp
        move.w   #0x2000,SR;       /* enable interrupts */
        jsr      _main
        bra      .;                /* loop in case main returns */
```

19.2.2 Boot Type Detection

Three GPIO lines define the boot mode and device type. After initialization of the on-chip resources, the boot loader reads the status of the GPIO lines and selects the requested device to boot from. Boot type encoding is described in Table 19-1.

Table 19-1. Boot Detection GPIO

Boot Mode	GPIO50	GPIO49	GPIO48
I^2C master	0	0	0
SPI (master)	0	0	1
IDE (master)	0	1	0
I^2C slave	1	0	0
UART (5.6448/11.2896 MHz Xtal)	1	0	1
UART (8.4672/16.9344/33.8688 MHz Xtal)	1	1	0
UART (5/10/20 MHz Xtal)	1	1	1

19.2.3 Serial Boot Data Format

All serial boot modes use the same data structure to read data from the external device. All data is organized in boot records. The boot loader reads one or more boot records and processes the data according to the command which is supplied in the boot record header. A boot record has the following structure:

Table 19-2. Boot Records

Offset	# Bytes	Description
0	1	Sync Byte (0x55)
1	1	Command/Data width (byte, word, longword)
2	4	Destination address in MCF5251 memory space
6	4	Number of bytes in data/code section (BC)
10	BC	Data/Code section

The first byte of a boot record is a sync byte with value 0x55. The boot-loader will ignore all data up to and including this byte. The second byte contains the command to execute and the data width of the data/code section. The command is coded in the upper nibble; the size is coded in the lower nibble. The size specification defines the word length to be used to store the data in the MCF5251 memory space. This allows writing to registers through the boot loader.

The following tables describe the encoding of the command and size bits:

Table 19-3. Command Bits

Command	Code
Store Immediate	0b0001
Execute	0b0011

Table 19-4. Size Bits

Size	Code
Byte	0b0001
Word	0b0010
Long Word	0b0100

19.2.3.1 Command Encoding/Size Encoding

The destination address is the base address for the data in the boot record. Data will be stored sequentially starting at this address.

The Byte Count (BC) is the number of data bytes in the boot record. After the loader has read the number of bytes specified, it assumes the next byte to be the start of a new record.

The data/code section in a command block contains the actual data to be written. This section can be empty (BC=0). Typically used to start program execution at the address specified in the address field.

19.2.3.2 Supported Commands

- Store Immediate
  - The store immediate command causes the boot loader to read from the serial device and store the data at the destination address as soon as a complete unit (1, 2 or 4 bytes) has been read. The unit size is defined in the lower nibble of the command word.
- Execute
  - The execute command has no data associated to it. The address field contains the entry point of the code to be executed. The byte count is 0. Upon reception of an execute command, the loader calls the routine at the specified address by means of a JSR instruction. If the called routine returns, the loader will continue and read the next boot record.

19.2.4 IDE Boot Data Format

An IDE boot record has the following structure:

Table 19-5. Boot Records

Offset	# Bytes	Description
0	4	Load address in MCF5251 memory space
4	4	Execution address
8	2	Boot record CRC
10	2	Length of record in sectors (max 256)
12	–	Data/Code section

The load address provides the start location where the boot record will be stored within the MCF5251's memory map, the execution address provides the entry point code execution will begin from after the boot record has been loaded into memory. The CRC allows the boot record to be validated before attempting to begin code execution. The length of the boot record is described in number of sectors (512 bytes) it occupies on the drive, up to a maximum of 256, making the maximum boot file size 131,060 bytes, 12 bytes are required for the boot record.

The boot record must be stored as the first three files in the root directory of a newly formatted FAT32 device. The files have to be in a continuous cluster (single section) and named as IDEBOOT1.IDE, IDEBOOT2.IDE and IDEBOOT3.IDE.

19.2.5 Boot Modes

19.2.5.1 Boot From I^2C / SPI – Master Mode

In I^2C master mode, the boot loader reads data from a serial EEPROM/FLASH connected to I2C0 (SCL0 and SDA0). The I^2C address of the device must be 0b1010000x, as this address is standard for serial memories. Only devices which use a 16-bit memory address are supported. The I^2C clock speed is set at 100KHz when the maximum crystal / external clock input is used (33.8688 MHz). The I^2C clock speed

will scale in a linear fashion with lower clock frequencies. Obviously the boot time will increase with lower frequencies, but the interface will still operate.

The loader uses ‘sequential read mode’ to retrieve the data, and starts reading from address zero.

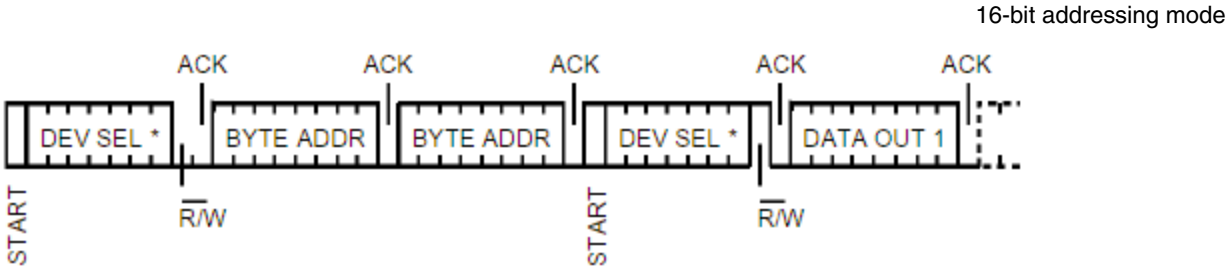


Figure 19-1. I²C Master Boot Mode

Boot from a serial device over the SPI interface is similar to the I²C master mode. In SPI mode the maximum bit rate is used, which is dependent on the clock input. QSPI_CS0, QSPI_DIN, QSPI_DOUT and QSPI_CLK provide the interface to the external device.

19.2.5.2 Boot from I²C - Slave Mode

In I²C slave mode, an external master can transfer boot records and data to the boot loader via I2C0, using the slave address 0b0101000x.

19.2.5.3 Boot from UART

In UART mode, the MCF5251 acts as a slave device and receives data over UART1 (TXD1 and RXD1).

UART configuration:

```
Baud rate:19200 / 9600 / 4800 baud @ Xtal = 33.8688 / 16.9344 / 8.4672 MHz (see config GPIO’s)
         19200 / 9600 baud @ Xtal = 11.2896 / 5.6448 MHz (see config GPIO’s)
         19200 / 9600 / 4800 baud @ Xtal = 20 / 10 / 5 MHz (see config GPIO’s)
Bits:    8
Parity   None
Stop Bits1
```

NOTE

The baud rate generator must be configured such that the actual baud rate is within +/- 3% of the intended baud rate.

19.2.5.3.1 UART Protocol

To allow a master device to synchronize with the MCF5251 during the boot process, the boot loader will echo all data received from the master. Data bytes are echoed as they are received. The master can use the echo to determine if the data has been received correctly or determine when an execute command has been completed.

19.2.5.4 Boot from IDE Device

The boot loader expects a partitioned disk, with the first partition being a FAT32 partition.

It will attempt to read the file IDEBOOT1.IDE and then verify the checksum. If this fails it tries IDEBOOT2.IDE eventually followed by IDEBOOT3.IDE.

Typically, the data in the boot file is a second stage boot loader which performs the system initialization (SDRAM etc) and loads the application code.

This second stage boot loader now performs the system boot, allowing the boot process to be customized, supporting different file systems, if necessary.

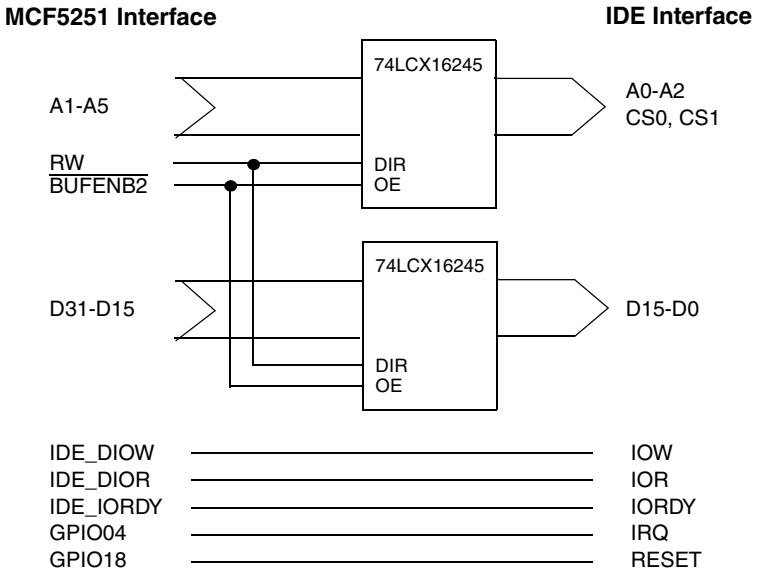


Figure 19-2. Boot Loader IDE Interface

19.3 Creating Appropriate Boot Record Files

For serial boot modes typically at least two boot record headers must be added to the raw binary file generated by the user code. The first must be at the start of the file and should be a ‘Store Immediate’ header with an appropriate load address and byte count, the second, an ‘Execute’ header, should be at the end of the file with an appropriate execute address.

Multiple ‘Store Immediate’ headers can be used if several separate blocks of data need to be loaded before code execution can begin.

Example utilities are available from Freescale to generate appropriate boot record files if required, contact your local Freescale representative for further details.

Chapter 20
Background Debug Mode (BDM) Interface

This chapter details the MCF5251 hardware debug support. The topics discussed are real-time trace support, background debug mode (BDM), and real-time debug support. The memory map, register descriptions and debug support operation are provided.

The MCF5251 implements an enhanced debug architecture. The original design plus these enhancements is known as Revision A (or Rev. A). The enhanced functionality is clearly identified in this chapter. The Rev. A enhancements are backward compatible with the original ColdFire debug definition.

The general topic of debug support is divided into three separate areas:

1. Real-Time Trace Support
2. Background Debug Mode (BDM)
3. Real-Time Debug Support

NOTE

To enable Debug Mode, TEST[2:0] pins must be = 001.

The logic required to support these three areas is contained in a debug module as shown in Figure 20-1.

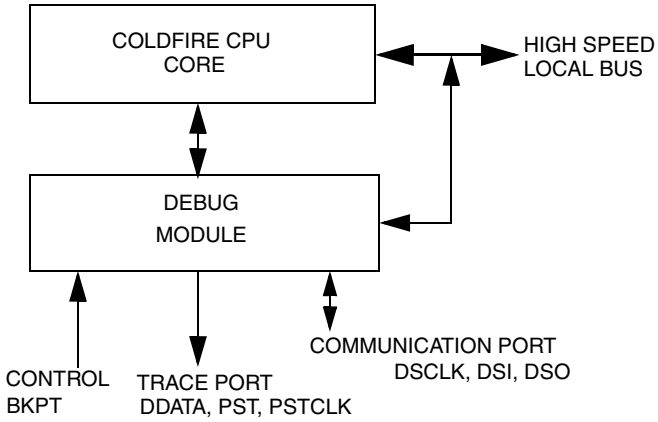


Figure 20-1. Processor/Debug Module Interface

20.1 Debug Support Signals

This section describes signals associated with the debug module. All ColdFire debug signals are unidirectional and are related to the rising-edge of the processor core's clock signal.

20.1.1 Breakpoint ($\overline{\text{BKPT}}$)

The $\overline{\text{BKPT}}$ active-low input signal is used to request a manual breakpoint. Its assertion causes the processor to enter a halted state after the completion of the current instruction. The halt status is reflected on the processor status (PST) pins as the value $F.

20.1.2 Debug Data (DDATA[3:0])

These output signals display the hardware 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 configuration/status register (CSR). Additionally, execution of the WDDATA instruction by the processor captures operands which are displayed on DDATA. These signals are updated each processor cycle.

20.1.3 Development Serial Clock (DSCLK)

This input signal is synchronized internally and provides the clock for the serial communication port to the debug module. The maximum frequency is 1/5 the speed of the processor's clock (CLK). At the synchronized rising edge of DSCLK, the data input on DSI is sampled, and the DSO output changes state. See Figure 20-3 for more information.

20.1.4 Development Serial Input (DSI)

The input signal is synchronized internally and provides the data input for the serial communication port to the debug module.

20.1.5 Development Serial Output (DSO)

This signal provides serial output communication for the debug module responses.

20.1.6 Processor Status (PST[3:0])

These output signals report the processor status. Table 20-1 shows the encoding of these signals. These outputs indicate the current status of the processor pipeline and are not related to the current bus transfer. The PST value is updated each processor cycle.

Table 20-1. Processor Status Encoding

PST[3:0]		Definition
(Hex)	(Binary)	
$0	0000	Continue execution
$1	0001	Begin execution of an instruction
$2	0010	Reserved
$3	0011	Entry into user-mode
$4	0100	Begin execution of **PULSE** and **WDDATA** instructions
$5	0101	Begin execution of taken branch or Sync_PC
$6	0110	Reserved
$7	0111	Begin execution of **RTE** instruction
$8	1000	Begin 1-byte transfer on DDATA
$9	1001	Begin 2-byte transfer on DDATA
$A	1010	Begin 3-byte transfer on DDATA
$B	1011	Begin 4-byte transfer on DDATA
$C	1100	Exception processing†
$D	1101	Emulator-mode entry exception processing†
$E	1110	Processor is stopped, waiting for interrupt†
$F	1111	Processor is halted †
Note: †These encodings are asserted for multiple cycles.		

20.1.7 Processor Status Clock (PSTCLK)

Since the debug trace port signals transition each processor cycle and are not related to the external bus frequency, an additional signal is output from the ColdFire microprocessor. The PSTCLK signal is a delayed version of the processor's high-speed clock and its rising-edge is used by the development system to sample the values on the PST and DDATA output buses. The PSTCLK signal is intended for use in the standard 26-pin debug connector. See Figure 20-39.

If the real-time trace functionality is not being used, the PCD bit of the CSR may be set (CSR[17] = 1) to force the PSTCLK, PST, and DDATA outputs to be disabled.

20.2 Real-Time Trace Support

In the area of debug functions, one fundamental requirement is support for real-time trace functionality. For example, definition of the dynamic execution path. 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 nibbles (4 bits): one nibble allows the processor to transmit information concerning the execution status of the core (processor status: PST), while the other nibble allows operand data to be

displayed. (debug data*:* DDATA). The processor status (PST) timing is synchronous with the processor status clock (PSTCLK) and may not be related to the current bus transfer. Table 20-1 shows the encoding of these signals.

The PST outputs can be used with an external image of the program to completely track the dynamic execution path of the machine when used with external development systems. The tracking of this dynamic path is complicated by any change-of-flow operation. This is especially evident when the branch target address is calculated based on the contents of a program-visible register (variant addressing.) For this reason, the DDATA outputs can be configured to display the target address of these types of change-of-flow instructions. Because the DDATA bus is only 4 bits wide, the address is displayed a nibble at a time across multiple clock cycles.

The debug module includes two 32-bit storage elements for capturing the internal ColdFire bus information. These two elements effectively form a FIFO buffer connecting the processor's high-speed local bus to the external development system through the DDATA signals. The FIFO buffer captures branch target addresses along with certain operand data values for eventual display on the DDATA output port, a nibble at a time, starting with the least-significant bit. The execution speed of the ColdFire processor is affected only when both storage elements contain valid data waiting to be dumped onto the DDATA port. In this case, the processor core is stalled until one FIFO entry is available. In all other cases, data output on the DDATA port does not impact execution speed.

20.2.1 Processor Status Signal Encoding

The PST signals are encoded to reflect the state of the Operand Execution Pipeline, and are generally not related to the current external bus transfer.

20.2.1.1 Continue Execution (PST = $0)

Many instructions complete in a single processor cycle. If an instruction requires more clock cycles, the subsequent clock cycles are indicated by driving the PST outputs with this encoding.

20.2.1.2 Begin Execution of an Instruction (PST = $1)

For most instructions, this encoding signals the first clock cycle of an instruction's execution. Certain change-of-flow opcodes, plus the PULSE and WDDATA instructions generate different encodings.

20.2.1.3 Entry into User Mode (PST = $3)

This encoding indicates the ColdFire processor has entered user mode. This encoding is signaled after the instruction which caused the user mode entry has executed.

20.2.1.4 Begin Execution of PULSE or WDDATA instructions (PST = $4)

The ColdFire instruction set architecture includes a PULSE opcode. This opcode generates a unique PST encoding, $4, when executed. This instruction can define logic analyzer triggers for debug and/or performance analysis. Additionally, a WDDATA instruction is supported that allows the processor core to write any operand (byte, word, longword) directly to the DDATA port, independent of any debug module

configuration. This opcode also generates the special PST encoding ($4) when executed, followed by the appropriate marker and then the data transfer on the DDATA outputs. The length of the data transfer is dependent on the operand size of the WDDATA instruction.

20.2.1.5 Begin Execution of Taken Branch (PST = $5)

This encoding is generated whenever a taken branch is executed. For certain opcodes, the branch target address may be optionally displayed on DDATA depending on the control parameters contained in the configuration/status register (CSR). The number of bytes of the address to be displayed is also controlled in the CSR and indicated by the PST marker value immediately preceding the DDATA outputs.

The bytes are always displayed in a least-significant to most-significant order. The processor captures only those target addresses associated with taken branches using a variant addressing mode. For example, all JMP and JSR instructions using address register indirect or indexed addressing modes, all RTE and RTS instructions as well as 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 these types of change-of-flow operations, the ColdFire processor uses the debug pins to output a sequence of information on successive processor clock cycles:

1. Identify a taken branch has been executed using the PST pins ($5).
2. Using the PST pins, optionally signal the target address is to be displayed on the DDATA pins. The encoding ($9, $A, $B) identifies the number of bytes that are displayed.
3. The new target address is optionally available on subsequent cycles using the nibble-wide DDATA port. The number of bytes of the target address displayed on this port is a configurable parameter (2, 3, or 4 bytes).

Another example of a variant branch instruction would be a JMP (A0) instruction. Figure 20-2 shows the outputs of the PST and DDATA signals when a JMP (A0) instruction executed, assuming the CSR is programmed to display the lower two bytes of an address.

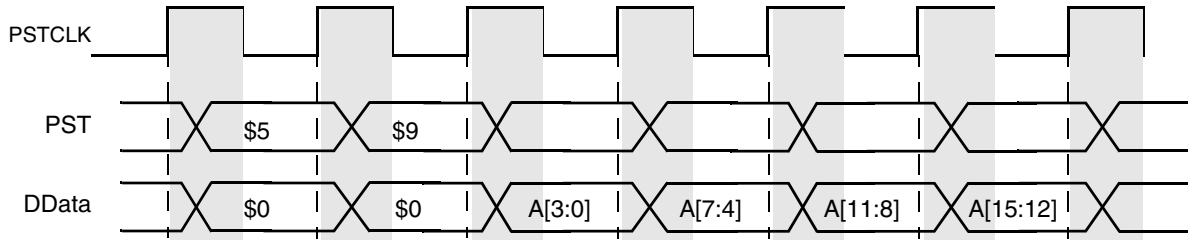


Figure 20-2. Example PST/DDATA Diagram

PST is driven with a $5 indicating a taken branch. In the second cycle, PST is driven with a marker value of $9 indicating a two-byte address that is displayed four bits at a time on the DDATA signals over the next four clock cycles. The remaining four clock cycles display the lower two-bytes of the address (A0), least significant nibble to most significant nibble. The output of the PST signals after the JMP instruction completes is dependent on the target instruction. The PST can continue with the next instruction before the address has completely displayed on the DDATA because of the DDATA FIFO. If the FIFO is full and the

next instruction needs to display captured values on DDATA, the pipeline stalls (PST = $0) until space is available in the FIFO.

20.2.1.6 Begin Execution of RTE Instruction (PST = $7)

The unique encoding is generated whenever the return-from-exception (RTE) instruction is executed.

20.2.1.7 Begin Data Transfer (PST = $8–$B)

These encodings serve as markers to indicate the number of bytes to be displayed on the DDATA port on subsequent clock cycles. This encoding is driven onto the PST port one processor cycle before the actual data is displayed on DDATA. When PST outputs a $8/$9/$A/$B marker value, the DDATA port outputs 1/2/3/4 bytes of captured data respectively on consecutive processor cycles.

20.2.1.8 Exception Processing (PST = $C)

This encoding is displayed during normal exception processing. Exceptions which enter emulation mode (debug interrupt, or optionally trace) generate a different encoding. Because this encoding defines a multicycle mode, the PST outputs are driven with this value until exception processing is completed.

20.2.1.9 Emulator Mode Exception Processing (PST = $D)

This encoding is displayed during emulation mode (debug interrupt, or optionally trace). Because this encoding defines a multicycle mode, the PST outputs are driven with this value until exception processing is completed.

20.2.1.10 Processor Stopped (PST = $E)

This encoding is generated as a result of the STOP instruction. The ColdFire processor remains in the stopped state until an interrupt occurs. Because this encoding defines a multicycle mode, the PST outputs are driven with this value until the stopped mode is exited.

20.2.1.11 Processor Halted (PST = $F)

This encoding is generated when the ColdFire processor is halted (Refer to Section 20.3.1, “CPU Halt.”) Because this encoding defines a multicycle mode, the PST outputs are driven with this value until the processor is restarted, or reset.

20.3 Background-Debug Mode (BDM)

Background debug mode (BDM) implements a low-level system debugger in the microprocessor hardware. Communication with the development system is handled through a dedicated, high-speed, full-duplex serial command interface. The BDM features are as follows:

- ColdFire implements the BDM controller in a dedicated hardware module. Although some BDM operations do require the CPU to be halted (For example, CPU register accesses), other BDM commands such as memory accesses can be executed while the processor is running.

- The read/write control register commands, RCREG and WCREG use the register coding scheme from the MOVEC instruction.
- The read/write debug module register commands, RDMREG and WDMREG support debug module register accesses.
- Illegal command responses can be returned using the FILL and DUMP commands, if not immediately preceded by certain, specific BDM commands.
- For any command performing a byte-sized memory read operation, the upper 8 bits of the response data are undefined. The referenced data is returned in the lower 8 bits of the response.
- The debug module forces alignment for memory-referencing operations: long accesses are forced to a 0-modulo-4 address; word accesses are forced to a 0-modulo-2 address. An address error response is never returned.

20.3.1 CPU Halt

Although many BDM operations can occur in parallel with CPU operation, unrestricted BDM operation requires the CPU to be halted. A number of sources can cause the CPU to halt, including the following as shown in order of priority:

1. The occurrence of the catastrophic fault-on-fault condition automatically halts the processor.
2. The occurrence of a hardware breakpoint can be configured to generate a pending halt condition in a manner similar to the assertion of the $\overline{\text{BKPT}}$ signal. In all cases, the assertion of this type of halt is first made pending in the processor. Next, the processor samples for pending halt and interrupt conditions once per instruction. Once the pending condition is asserted, the processor halts execution at the next sample point. See Section 20.4.1, “Theory of Operation,” for more detail.
3. The execution of the HALT ColdFire instruction immediately suspends execution. By default this is a supervisor instruction and attempted execution while in user mode generates a privilege violation exception. A User Halt Enable (UHE) control bit is provided in the Configuration/Status Register (CSR) to allow execution of HALT in user mode. The processor may be restarted after the execution of the HALT instruction by serial shifting a “GO” command into the debug module. Execution continues at the instruction following the HALT opcode.
4. The assertion of the $\overline{\text{BKPT}}$ input pin is treated as a pseudo-interrupt. For example, the halt condition is made pending until the processor core samples for halts/interrupts. The processor samples for these conditions once during the execution of each instruction. If there is a pending halt condition at the sample time, the processor suspends execution and enters the halted state.

There are two special cases involving the assertion of the $\overline{\text{BKPT}}$ pin to be considered.

After the system reset signal is negated, the processor waits for 16 clock cycles before beginning reset exception processing. If the $\overline{\text{BKPT}}$ input pin is asserted within the first eight cycles after $\overline{\text{RSTI}}$ is negated, the processor enters the halt state, signaling that halt status, ($F), on the PST outputs. While in this state, all resources accessible through the debug module can be referenced. This is the only opportunity to force the ColdFire processor into emulation mode using the EMU bit in the configuration/status register (CSR). Once the system initialization is complete, the processor response to a BDM GO command is dependent on the set of BDM commands performed while breakpointed. Specifically, if the processor’s PC register

was loaded, then the GO command simply causes the processor to exit the halted state and pass control to the instruction address contained in the PC.

NOTE

In this case, the normal reset exception processing is bypassed. Conversely, if the PC register was not loaded, then the GO command causes the processor to exit the halted state and continue with reset exception processing.

ColdFire also handles a special case of the assertion of $\overline{\text{BKPT}}$ 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. Once halted, all BDM commands may be exercised. When the processor is restarted, it continues with the execution of the next sequential instruction. For example, the instruction following the STOP opcode.

The halt source is indicated in CSR[27:24]. For simultaneous halt conditions, the highest priority source is indicated.

20.3.2 BDM Serial Interface

Once the CPU is halted and the halt status reflected on the PST outputs, the development system may send unrestricted commands to the debug module. The debug module implements a synchronous protocol using a three-pin interface: development serial clock (DSCLK), development serial input (DSI), and development serial output (DSO). The development system serves as the serial communication channel master and is responsible for generation of the clock (DSCLK). The maximum operating bandwidth of the serial channel is DC to 1/5 of the processor frequency. The channel uses a full duplex mode, where data is transmitted and received simultaneously by both 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 20-3, all state transitions are enabled on a rising edge of the processor clock when DSCLK is high. For example, DSI is sampled and DSO is driven. The DSCLK signal must also be sampled low (on a positive edge of CPUCLK) between each bit exchange. The MSB is transferred first.

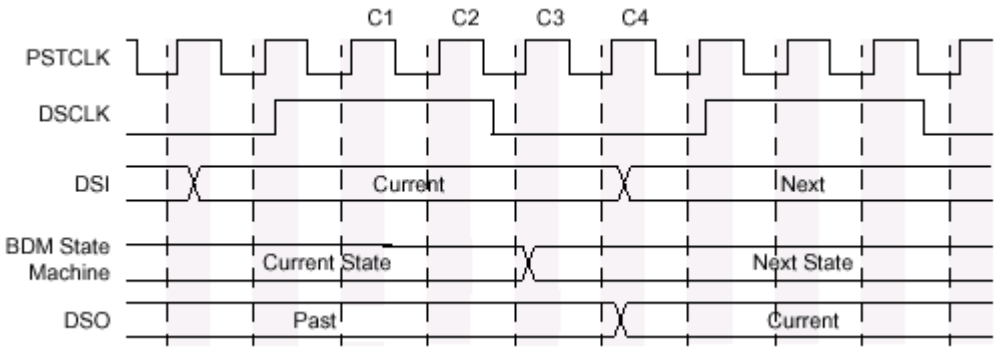


Figure 20-3. BDM Serial Transfer

Both DSCLK and DSI are synchronized inputs.The DSCLK signal essentially acts as a pseudo “clock enable” and is sampled on the rising edge of CPUCLK as well as the DSI. The DSO output is delayed from the DSCLK-enabled CPUCLK rising edge. All events in the debug module’s serial state machine are based on the rising edge of the microprocessor clock.

20.3.2.1 Receive Packet Format

The basic receive packet of information is 17 bits long,16 data bits plus a status bit, as shown in Figure 20-4.

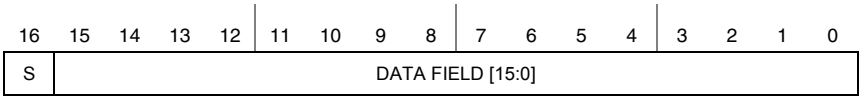


Figure 20-4. Receive BDM Packet Register

Table 20-2 describes the receive BDM packets. Bit descriptions are described in Table 20-3.

Table 20-2. CPU-Generated Command Responses

S Bit	Data	Message Type
0	xxxx	Valid data transfer
0	$FFFF	Status OK
1	$0000	Not ready with response; try again
1	$0001	Error—terminated bus cycle; data invalid
1	$FFFF	Illegal command

Table 20-3. Receive BDM Packet Register Field Descriptions

Field	Description
16 S-Status	The status bit indicates the status of CPU-generated messages as shown in Table 20-2.
15–0 Data	The data field contains the message data to be communicated from the debug module to the development system. The response message is always a single word, with the data field encoded as shown in Table 20-2.

20.3.2.2 Transmit Packet Format

The basic transmit packet of information is 17 bits long,16 data bits plus a control bit, as shown in Figure 20-5.

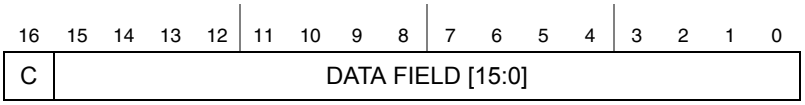


Figure 20-5. Transmit BDM Packet Register

Table 20-4. Transmit BDM Packet Register Field Descriptions

Field	Description
16 C-Control	The Control Bit (Bit 16) is reserved. Command and data transfers initiated by the development system should clear bit 16.
15–0 Data Field	The data field contains the message data to be communicated from the development system to the debug module.

20.3.3 BDM Command Set

ColdFire supports a subset of BDM commands to provide access to new hardware features. The BDM commands must not be issued when the ColdFire processor is accessing the debug module registers using the WDEBUG instruction, or the resulting behavior is undefined.

20.3.3.1 BDM Command Set Summary

The BDM command set is summarized in Table 20-5. Subsequent sections contain detailed descriptions of each command.

Table 20-5. BDM Command Summary

Command	Mnemonic	Description	CPU Impact[1]	Command (HEX)	Section Page
READ A/D REGISTER	RAREG/RDREG	Read the selected address or data register and return results through the serial interface.	Halted	$218 {A/D, Reg[2:0]}	20.3.4.1.1/20-13
WRITE A/D REGISTER	WAREG/WDREG	The data operand is written to the specified address or data register.	Halted	$208 {A/D, Reg[2:0]}	20.3.4.1.2/20-14
READ MEMORY LOCATION	READ	Read the data at the memory location specified by the longword address.	Steal	$1900–byte $1940–wd $1980–long	20.3.4.1.3/20-14
WRITE MEMORY LOCATION	WRITE	Write the operand data to the memory location specified by the longword address.	Steal	$1800–byte $1840–wd $1880–long	20.3.4.1.4/20-16
DUMP MEMORY BLOCK	DUMP	Used with the READ command to dump large blocks of memory. An initial READ is executed to set up the starting address of the block and to retrieve the first result. Subsequent operands are retrieved with the DUMP command.	Steal	$1D00–byte $1D40–wd $1D80–long	20.3.4.1.5/20-17
FILL MEMORY BLOCK	FILL	Used with the WRITE command to fill large blocks of memory. An initial WRITE is executed to set up the starting address of the block and to supply the first operand. Subsequent operands are written with the FILL command.	Steal	$1C00–byte $1C40–word $1C80–long	20.3.4.1.6/20-19
RESUME EXECUTION	GO	The pipeline is flushed and refilled before execution resumes at the current PC.	Halted	$0C00	20.3.4.1.7/20-21
NO OPERATION	NOP	NOP performs no operation and may be used as a null command.	Parallel	$0000	20.3.4.1.8/20-21
READ CONTROL REGISTER	RCREG	Read the system control register.	Halted	$2980	20.3.4.1.9/20-22
WRITE CONTROL REGISTER	WCREG	Write the operand data to the system control register.	Halted	$2880	20.3.4.1.10/20-23
READ DEBUG MODULE REGISTER	RDMREG	Read the debug module register.	Parallel	$2D {$4† DRc[4:0]}	20.3.4.1.11/20-24
WRITE DEBUG MODULE REGISTER	WDMREG	Write the operand data to the debug module register.	Parallel	$2C {$4† Drc[4:0]}	20.3.4.1.12/20-24

[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 which can be interleaved with CPU accesses
Parallel—Command is executed in parallel with CPU activity
Refer to command summaries for detailed operation descriptions.

All ColdFire Family BDM commands include a 16-bit operation word followed by an optional set of one or more extension words as shown in Table 20-6.

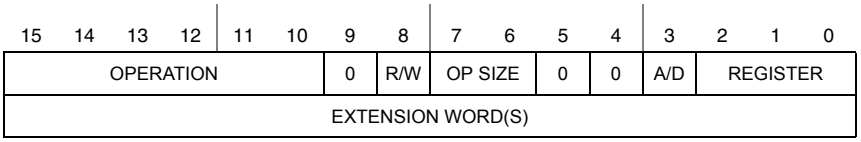


Figure 20-6. BDM Command Register

Table 20-6 describes the BDM fields.

Table 20-6. BDM Command Register Field Descriptions

Field	Description
Operation	The operation field specifies the command.
R/W	The R/W field specifies the direction of operand transfer. When the bit is set, the transfer is from the CPU to the development system. When the bit is cleared, data is written to the CPU or to memory from the development system.
Operand Size	For sized operations, this field specifies the operand data size. All addresses are expressed as 32-bit absolute values. The size field is encoded as listed in Table 20-7.
Address/Data (A/D)	The A/D field is used in commands that operate on address and data registers in the processor. It determines whether the register field specifies a data or address register. A one indicates an address register; zero, a data register.
Register	In commands that operate on processor registers, this field specifies which register is selected. The field value contains the register number.
Extension Word(s) (as required)	Certain commands require extension words for addresses and/or immediate data. Addresses require two extension words because only absolute long addressing is permitted. Immediate data can be either one or two words in length—byte and word data each require a single extension word; longword data requires two words. Both 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.”

Table 20-7. BDM Size Field Encoding

Encoding	Operand Size	Bit Values
00	Byte	8 bits
01	Word	16 bits
10	Longword	32 bits
11	Reserved	

20.3.4 Command Sequence Diagram

A command sequence diagram (see Figure 20-7) shows the serial bus traffic for each command. Each bubble in the diagram represents a single 17-bit transfer across the bus. The top half in each bubble corresponds to the data transmitted by the development system to the debug module; the bottom half corresponds to the data returned by the debug module in response to the previous development system commands. Command and result transactions are overlapped to minimize latency.

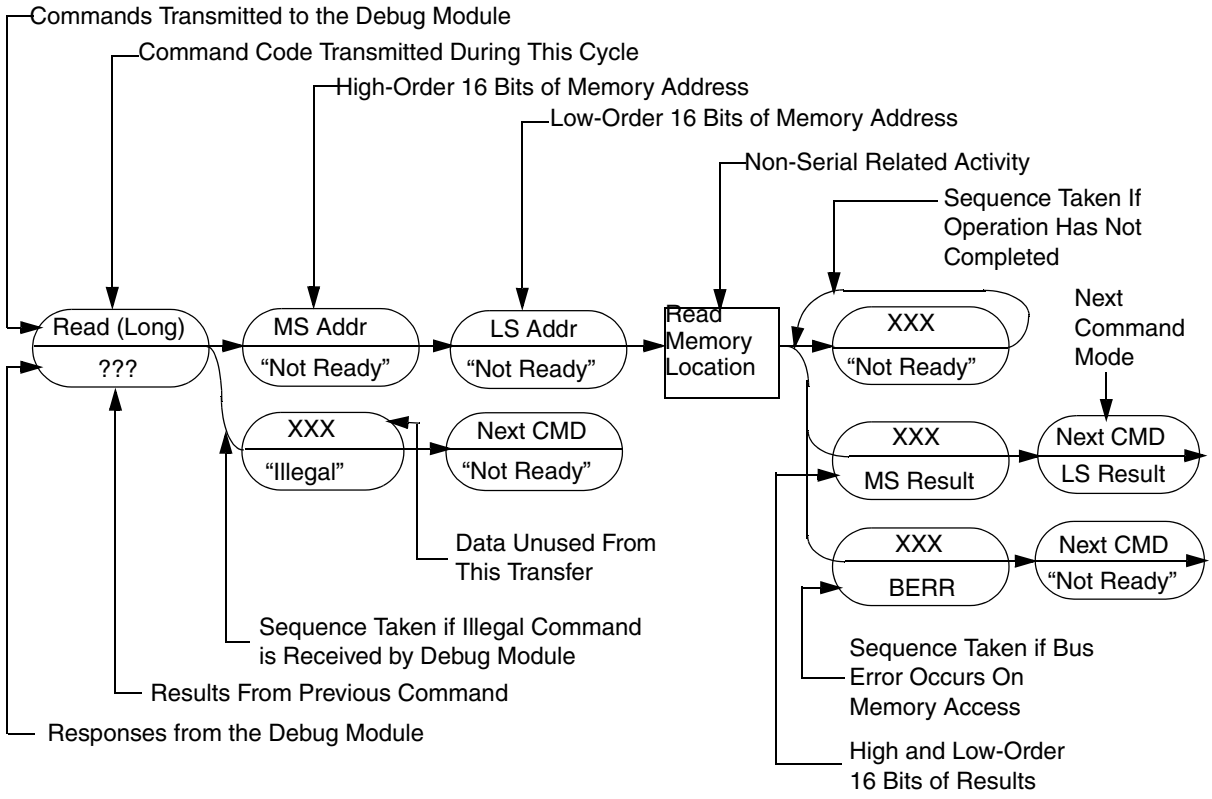


Figure 20-7. Command Sequence Diagram

The cycle in which the command is issued contains the development system command mnemonic (in this example, "read memory location"). During the same cycle, the debug module responds with either the low-order results of the previous command or a command complete status (if no results were required).

During the second cycle, the development system supplies the high-order 16 bits of the memory address. The debug module returns a "not ready" response unless the received command was decoded as unimplemented, in which case the response data is the illegal command encoding. If an illegal command response occurs, the development system should retransmit the command.

NOTE

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.

In the third cycle, the development system supplies the low-order 16 bits of a memory address. The debug module always returns the "not ready" response in this cycle. At the completion of the third cycle, the

debug module initiates a memory read operation. Any serial transfers that begin while the memory access is in progress return the “not ready” response.

Results are returned in the two serial transfer cycles following the completion of the memory access. The data transmitted to the debug module during the final transfer is the opcode for the following command. If a memory or register access is terminated with a bus error, the error status (S=1, DATA=$0001) is returned in place of the result data.

20.3.4.1 Command Set Descriptions

The BDM command set is summarized in Table 20-5. Subsequent sections contain detailed descriptions of each command.

NOTE

The BDM status bit (S) is zero for normally-completed commands, while illegal commands, “not ready” responses and bus-error transfers return a logic one in the S bit. Refer to Section 20.3.2, “BDM Serial Interface,” for information on the serial packet receive packet format.

Unassigned command opcodes are reserved by Freescale for future expansion. All unused command formats within any revision level perform a NOP and return the ILLEGAL command response.

20.3.4.1.1 Read Address/Data Register (RAREG/RDREG)

RAREG and RDREG reads 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.

	15–12	11–8	7–4	3	2–0
Command	0x2	0x1	0x8	A/D	Register
Result	D[31:16]				
	D[15:0]				

Figure 20-8. Command/Result Formats

Command Sequence:

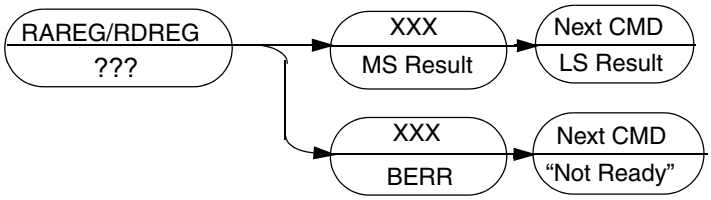


Figure 20-9. Read A/D Register Command Sequence

Operand Data:

None

Result Data:

The contents of the selected register are returned as a longword value. The data is returned most significant word first.

20.3.4.1.2 Write Address/Data Register (WAREG and WDREG)

WAREG and WDREG write the operand longword data to the specified address or data register. All 32 register bits are altered by the write. A bus error response is returned if the CPU core is not halted.

Command Format:

Table 20-8. WAREG/WDREG Command

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$2				$0				$8				A/D	REGISTER		
DATA [31:16]															
DATA [15:0]															

Command Sequence:

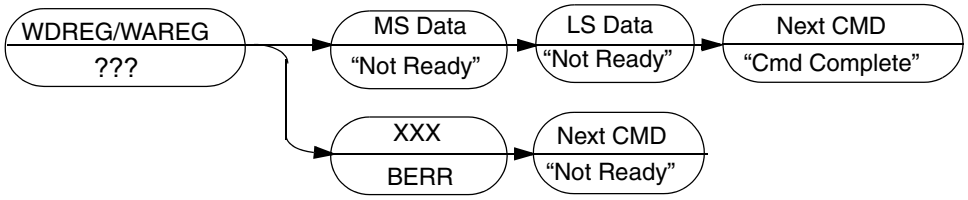


Figure 20-10. Write A/D Register Command Sequence

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 the data $FFFF (with the status bit cleared) when the register write is complete.

20.3.4.1.3 Read Memory Location (READ)

The READ command reads the operand data from the memory location specified by the longword address. The address space is defined by the contents of the low-order 5 bits {TT, TM} of the BDM Address Attribute Register (BAAR). The hardware forces the low-order bits of the address to zeros for word and longword accesses to ensure that operands are always accessed on natural boundaries: words on 0-modulo-2 addresses, longwords on 0-modulo-4 addresses.

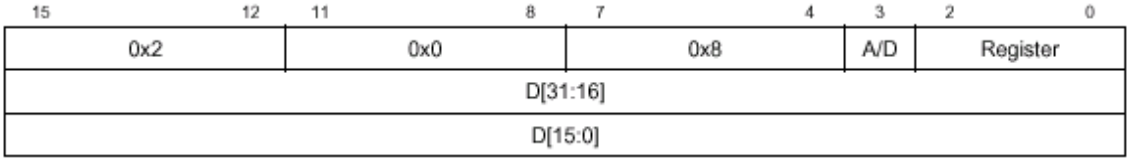


Figure 20-11. WAREG/WDREG Command Format

		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 20-12. READ Command/Result Format

Command Sequence:

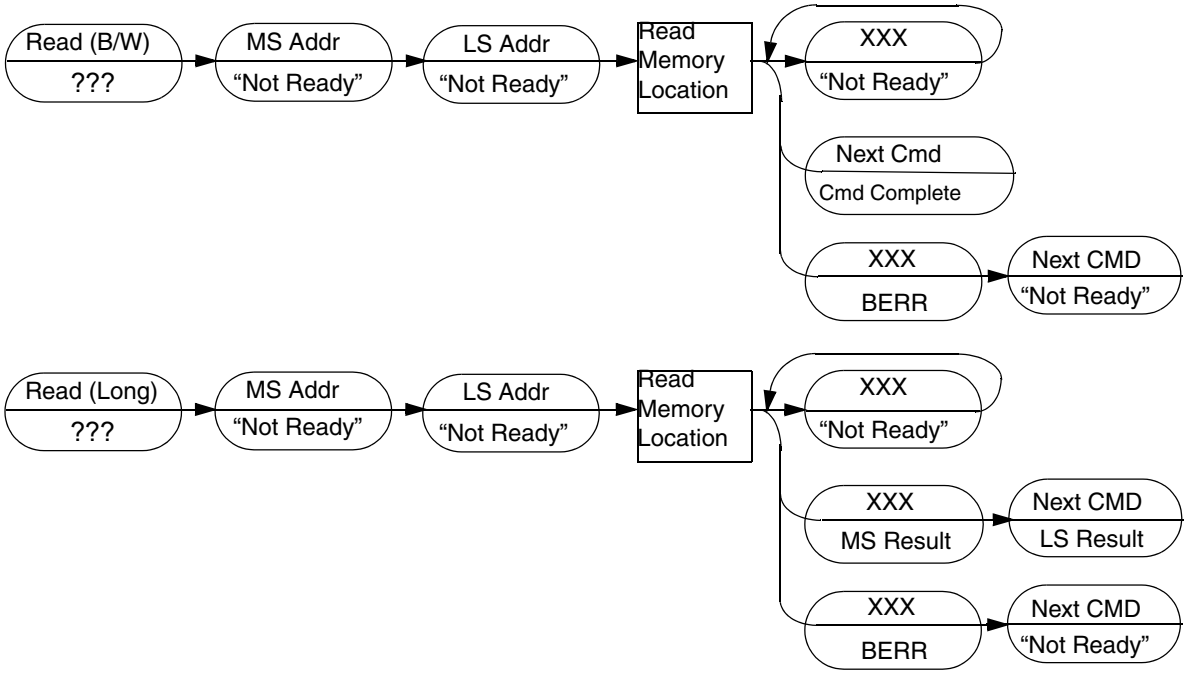


Figure 20-13. Read Memory Location Command Sequence

Operand Data:

The single operand is the longword address of the requested memory location.

Result Data:

The requested data is returned as either a word or longword. Byte data is returned in the least significant byte of a word result, with the upper byte undefined. Word results return 16 bits of significant data; longword results return 32 bits. A value of $0001 (with the status bit set) is returned if a bus error occurs.

20.3.4.1.4 Write Memory Location (WRITE)

The WRITE command writes the operand data to the memory location specified by the longword address. The address space is defined by the contents of the low-order 5 bits {TT, TM} of the BDM Address Attribute Register (BAAR). The hardware forces the low-order bits of the address to zeros for word and longword accesses to ensure that operands are always accessed on natural boundaries: words on 0-modulo-2 addresses, longwords on 0-modulo-4 addresses.

Command Sequence:

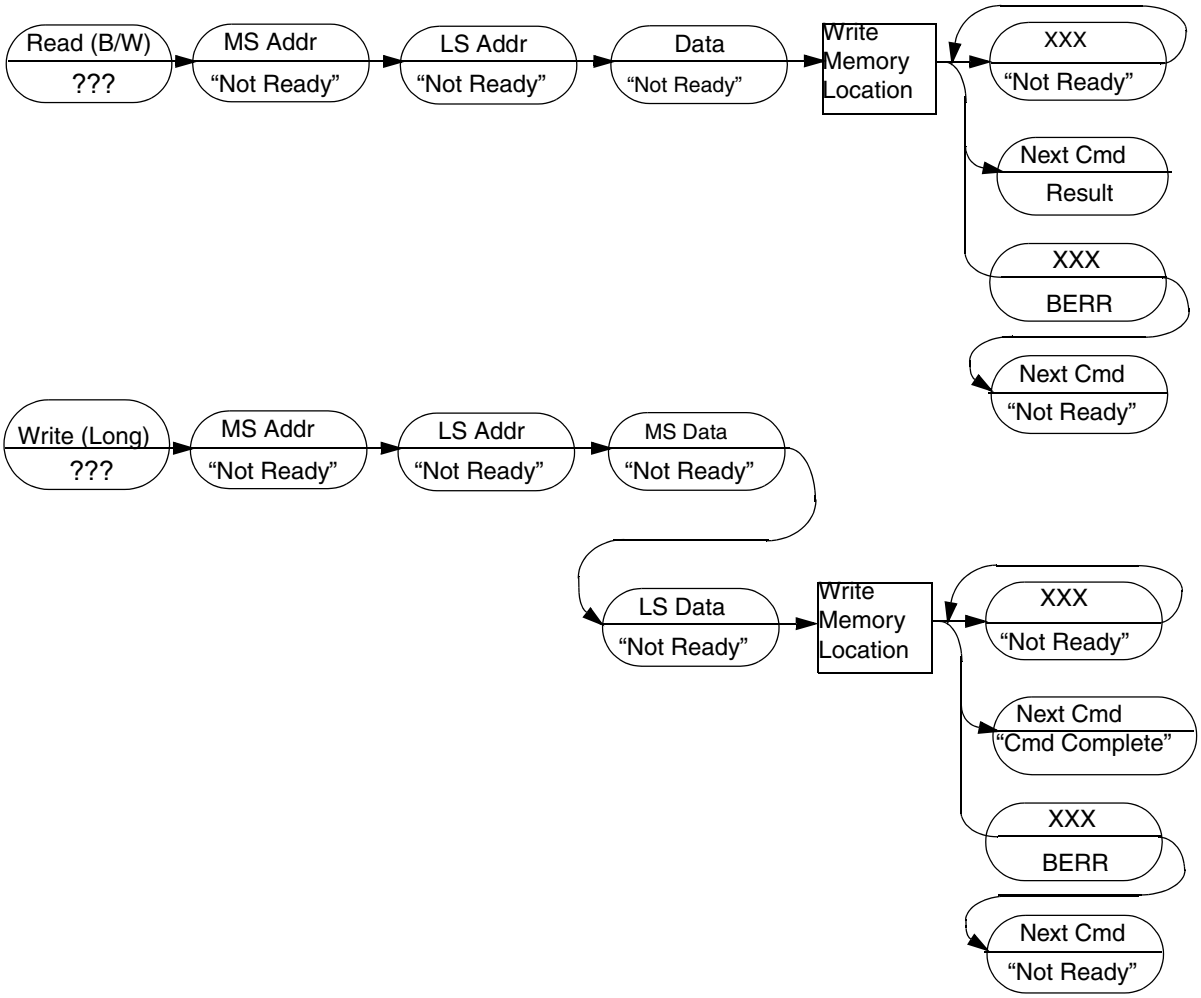


Figure 20-14. Write Memory Location Command Sequence

Operand Data:

Two operands are required for this instruction. The first operand is a longword absolute address that specifies a location to which the operand data is to be written. The second operand is the data. Byte data is transmitted as a 16-bit word, justified in the least significant byte; 16- and 32-bit operands are transmitted as 16 and 32 bits, respectively.

Result Data:

Command complete status is indicated by returning the data $FFFF (with the status bit cleared) when the register write is complete. A value of $0001 (with the status bit set) is returned if a bus error occurs.

20.3.4.1.5 Dump Memory Block (DUMP)

DUMP is used in conjunction 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. The DUMP command retrieves subsequent operands. The initial address is incremented by the operand size (1, 2, or

4) and saved 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

The DUMP command does not check for a valid address. DUMP is a valid command only when preceded by another DUMP, NOP, or by a READ 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 DUMP command is processed, allowing the operand size to be dynamically altered.

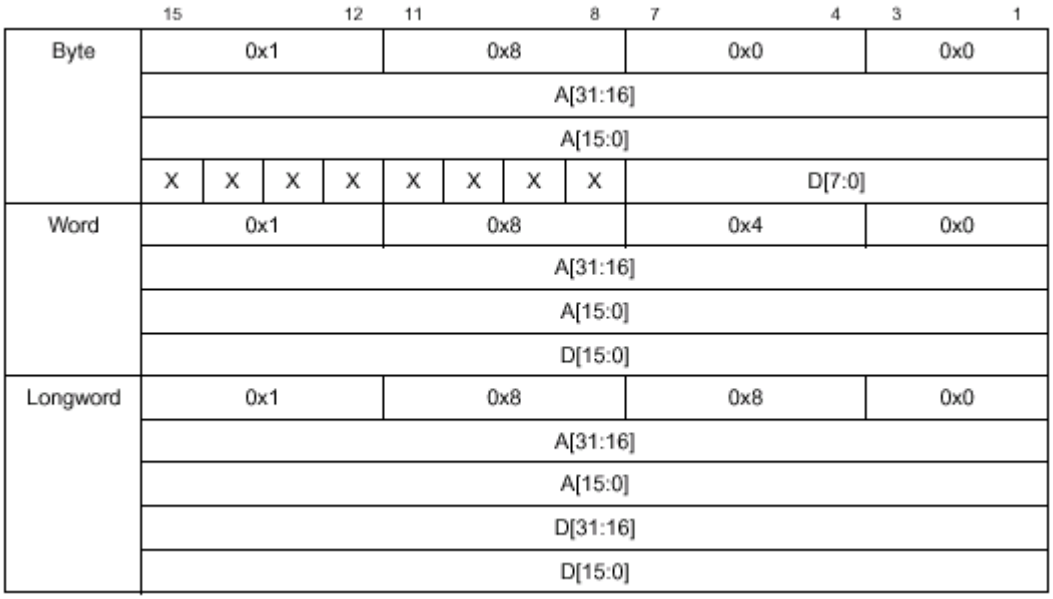


Figure 20-15. DUMP Command/Result Format

Command Sequence:

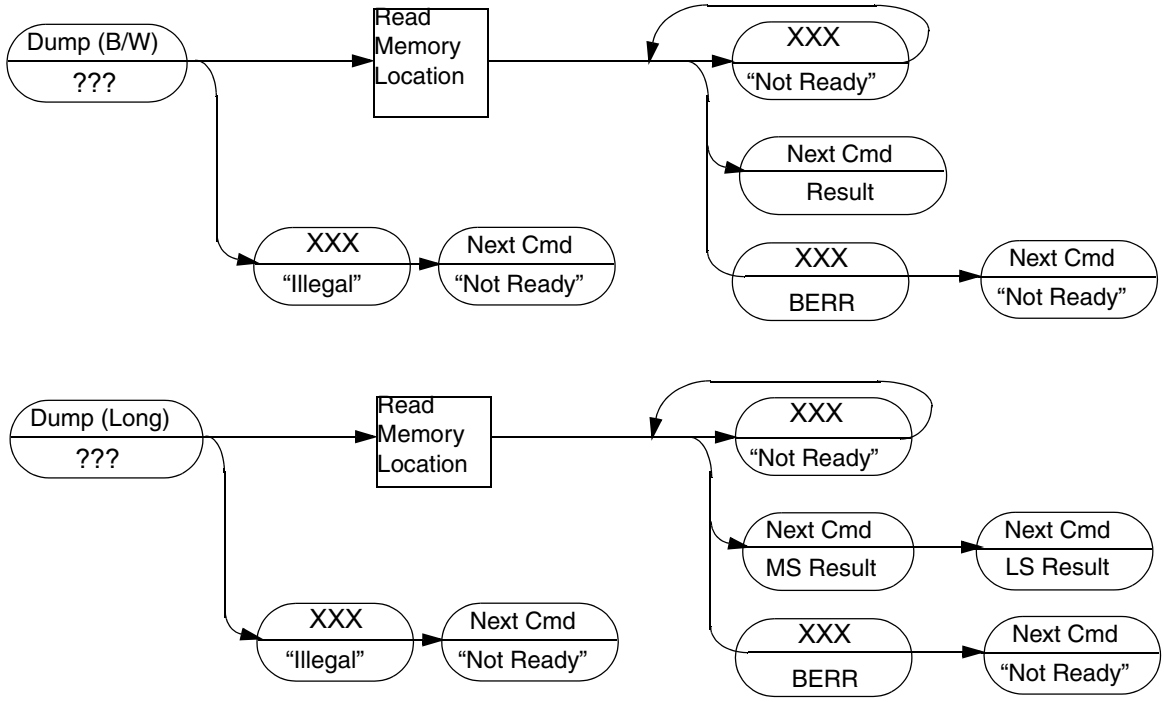


Figure 20-16. DUMP Memory Block Command Sequence

Operand Data:

None

Result Data:

Requested data is returned as either 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 $0001 (with the status bit set) is returned if a bus error occurs.

20.3.4.1.6 Fill Memory Block (FILL)

FILL is used in conjunction 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 is incremented by the operand size (1, 2, or 4) and saved 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.

NOTE

The FILL command does not check for a valid address FILL is a valid command only when preceded by another FILL, NOP or by 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:

Table 20-9. Byte FILL Command

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$1				$C				$0				$0			
X	X	X	X	X	X	X	X	DATA [7:0]							

Table 20-10. Word FILL Command

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$1				$C				$4				$0			
DATA [15:0]															

Table 20-11. Long FILL Command

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$1				$C				$8				$0			
DATA [31:16]															
DATA [15:0]															

Command Sequence:

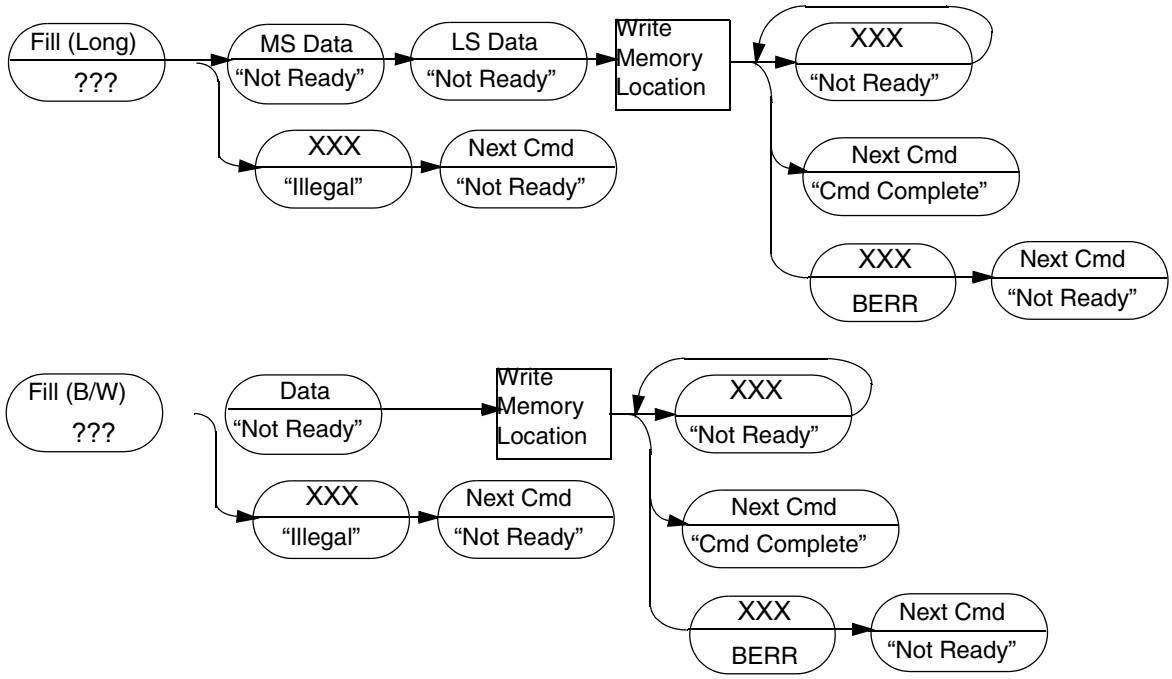


Figure 20-17. Fill Memory Block Command Sequence

Operand Data:

A single operand is data to be written to the memory location. Byte data is transmitted as a 16-bit word, justified in the least significant byte; 16- and 32-bit operands are transmitted as 16 and 32 bits, respectively.

Result Data:

Command complete status is indicated by returning the data $FFFF (with the status bit cleared) when the register write is complete. A value of $0001 (with the status bit set) is returned if a bus error occurs.

20.3.4.1.7 Resume Execution (GO)

The GO command flushes and refills the pipeline before resuming normal instruction execution. Prefetching begins at the current PC and current privilege level. If any register (For example, the PC or SR) was altered by a BDM command while halted, the updated value is used as the prefetching resumes.

Command Formats:

Table 20-12. GO Command

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$0				$C				$0				$0			

Command Sequence:

Figure 20-18. Resume Execution

Operand Data:

None

Result Data:

The “command complete” response ($0FFFF) is returned during the next shift operation.

20.3.4.1.8 No Operation (NOP)

NOP performs no operation and may be used as a null command where required.

Command Formats:

Table 20-13. NOP Command

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$0				$0				$0				$0			

Command Sequence:

Figure 20-19. No Operation Command Sequence

Operand Data:

None

Result Data:

The “command complete” response, $FFFF (with the status bit cleared), is returned during the next shift operation.

20.3.4.1.9 Read Control Register (RCREG)

RCREG reads the selected control register and returns the 32-bit result. Accesses to the processor/memory control registers are always 32 bits in size, regardless of the implemented register width. The second and third words of the command effectively form a 32-bit address used by the debug module to generate a special bus cycle to access the specified control register. The 12-bit Rc field is the same as that used by the MOVEC instruction.

	15–12	11–8	7–4	3–0
Command	0x2	0x9	0x8	0x0
	0x0	0x0	0x0	0x0
	0x0	Rc		
Result	D[31:16]			
	D[15:0]			

Figure 20-20. RCREG Command/Result Formats

Rc encoding:

Table 20-14. Control Register Map

Rc	Register Definition
$002	Cache Control Register (CACR)
$004	Access Control Register 0 (ACR0)
$005	Access Control Register 1 (ACR1)
$801	Vector BASE Register (VBR)
$804	MAC Status Register (MACSR)
$805	MAC Mask Register (MASK)
$806	MAC Accumulator (ACC0)
$807	MAC Accumulator (ACC1)
$808	MAC Accumulator (ACC2)
$80B	MAC Accumulator (ACC3)
$80E	Status Register (SR)
$80F	Program Register (PC)
$C04	RAM Base Address Register (RAMBAR0)
$C05	RAM Base Address Register (RAMBAR1)

Table 20-14. Control Register Map

Rc	Register Definition
$C0F	Module Base Address Register (MBAR)
$C0E	Module Base Address Register (MBAR2)

20.3.4.1.10 Write Control Register (WCREG)

The operand (longword) data is written to the specified control register. The write alters all 32 register bits.

	15–12	11–8	7–4	3–0
Command	0x2	0x8	0x8	0x0
	0x0	0x0	0x0	0x0
	0x0	Rc		
Result	D[31:16]			
	D[15:0]			

Figure 20-21. WCREG Command Sequence

Command Sequence:

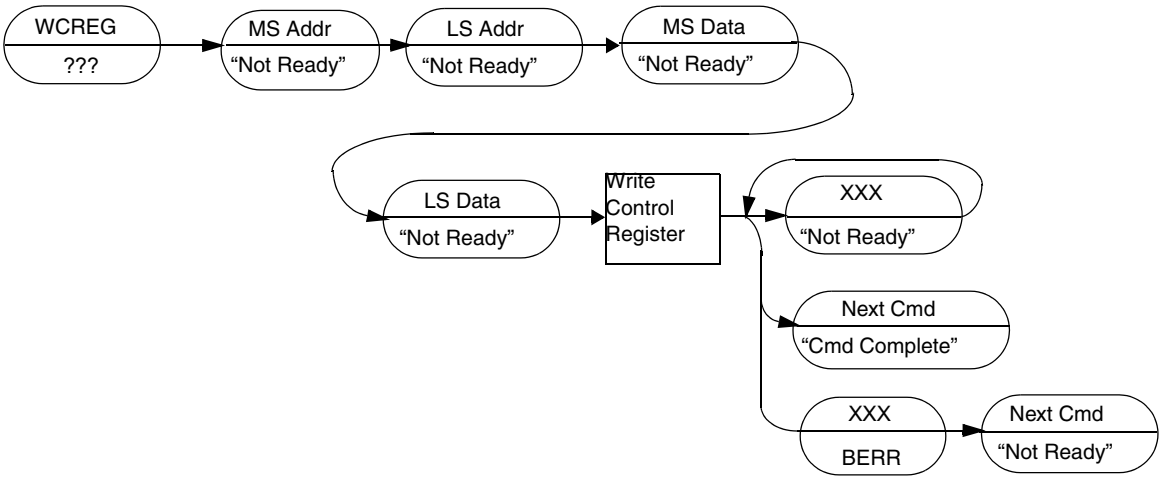


Figure 20-22. Write Control Register Command Sequence

Operand Data:

Two operands are required for this instruction. The first long operand selects the register to which the operand data is to be written. The second operand is the data.

Result Data:

Successful write operations return a $FFFF. Bus errors on the write cycle are indicated by the assertion of bit 16 in the status message and by a data pattern of $0001.

20.3.4.1.11 Read Debug Module Register (RDMREG)

RDMREG reads the selected debug module register and return the 32-bit result. The only valid register selection for the RDMREG command is the CSR (DRc = $0).

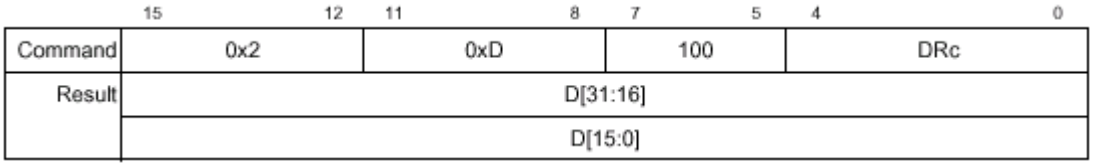


Figure 20-23. RDMREG Command/Result Register

DRc encoding:

Table 20-15. Definition of DRc Encoding—Read

DRc[3:0]	Debug Register Definition	Mnemonic	Initial State
$0	Configuration/Status	CSR	$0
$1-$F	Reserved	–	–

Command Sequence:

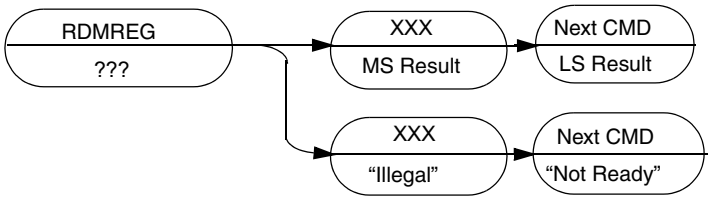


Figure 20-24. Read Debug Module Register Command Sequence

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.

20.3.4.1.12 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. The DSCLK signal must be inactive while debug module register writes from the CPU accesses are performed using the WDEBUG instruction.

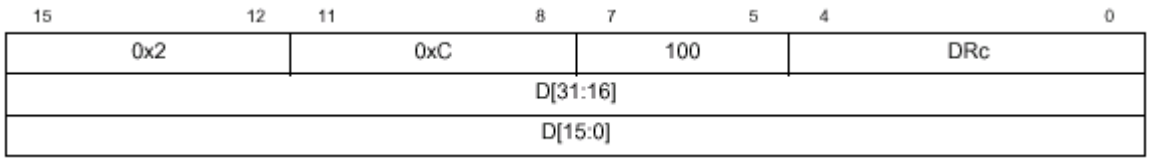


Figure 20-25. WDMREG BDM Command Register

DRc encoding:

Table 20-16. Definition of DRc Encoding—Write

DRc[3:0]	Debug Register Definition	Mnemonic	Initial State
$0	Configuration/Status	CSR	$0
$1-$4	Reserved	–	–
$5	BDM Address Attribute	BAAR	$5
$6	Bus Attributes and Mask	AATR	$5
$7	Trigger Definition	TDR	$0
$8	PC Breakpoint	PBR	–
$9	PC Breakpoint Mask	PBMR	–
$A-$B	Reserved	–	–
$C	Operand Address High Breakpoint	ABHR	–
$D	Operand Address Low Breakpoint	ABLR	–
$E	Data Breakpoint	DBR	–
$F	Data Breakpoint Mask	DBMR	–

Command Sequence:

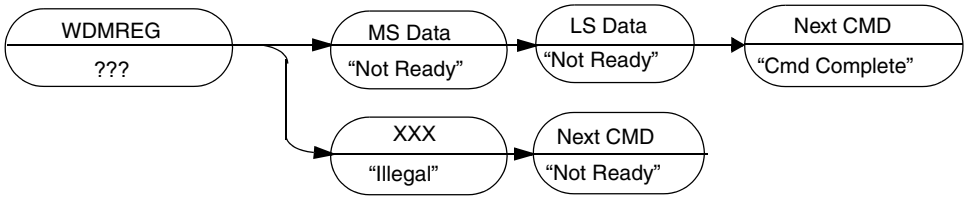


Figure 20-26. Write Debug Module Register Command Sequence

Operand Data:

Longword data is written into the specified debug register. The data is supplied most significant word first.

Result Data:

Command complete status ($0FFFF) is returned when register write is complete.

20.3.4.1.13 Unassigned Opcodes

Unassigned command opcodes are reserved by Freescale. All unused command formats within any revision level perform a NOP and return the ILLEGAL command response.

20.3.4.2 BDM Accesses of the eMAC Registers

The presence of rounding logic in the output data path 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 (ACCx) requires the following sequence:

```
Bdm Read ACCx (
      rcreg macsr; // read current macsr contents & 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:

```
Bdm Write ACCx (
      rcreg macsr; // read current macsr contents & 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.

Command Sequence:

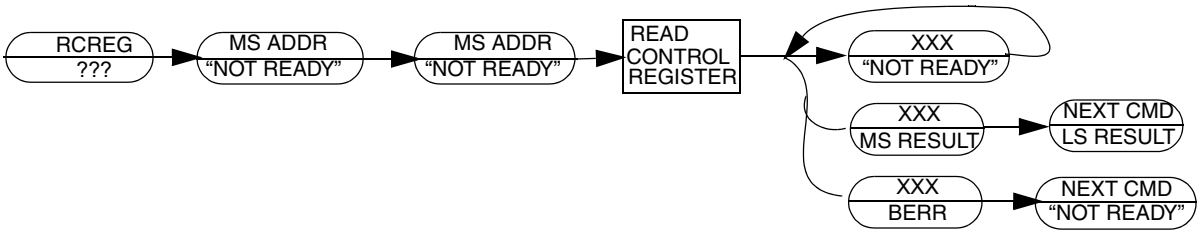


Figure 20-27. Read Control Register Command Sequence

Operand Data:

The single operand is the 32-bit Rc control register select field.

Result Data:

The contents of the selected control register are returned as a longword value. The data is returned most significant word first. For those control registers with widths less than 32 bits, only the implemented portion of the register is guaranteed to be correct. The remaining bits of the longword are undefined.

20.4 Real-Time Debug Support

The ColdFire Family provides support for the debug of real-time applications. For these types of embedded systems, the processor cannot be halted during debug, but must continue to operate. The foundation of this area of debug support is that while the processor cannot be halted to allow debugging, the system can generally tolerate small intrusions into the real-time operation.

The debug module provides a number of hardware resources to support various hardware breakpoint functions. Specifically, three types of breakpoints are supported: PC with mask, operand address range, and data with mask. These three basic breakpoints can be configured into one- or two-level triggers with

the exact trigger response also programmable. The debug module programming model is accessible from either the external development system using the serial interface or from the processor's supervisor programming model using the WDEBUG instruction.

20.4.1 Theory of Operation

The breakpoint hardware can be configured to respond to triggers in several ways. The desired response is programmed into the Trigger Definition Register (TDR). In all situations where a breakpoint triggers, an indication is provided on the DDATA output port, when not displaying captured operands or branch addresses, as shown in Table 20-17.

Table 20-17. DDATA[3:0], CSR[31:28] Breakpoint Response

DDATA[3:0], CSR[31:28]	Breakpoint Status
$000x	No Breakpoints Enabled
$001x	Waiting for Level 1 Breakpoint
$010x	Level 1 Breakpoint Triggered
$101x	Waiting for Level 2 Breakpoint
$110x	Level 2 Breakpoint Triggered
All other encodings are reserved for future use.	

The breakpoint status is also posted in the CSR.

The BDM instructions load and configure the desired breakpoints using the appropriate registers. As the system operates, a breakpoint trigger generates a response as defined in the TDR. If the system can tolerate the processor being halted, a BDM-entry can be used. With the TRC bits of the TDR equal to $1, the breakpoint trigger causes the core to halt as reflected in the PST = $F status.

NOTE

For PC breakpoints, the halt occurs before the targeted instruction is executed. For address and data breakpoints, the processor may have executed several additional instructions. As a result, trigger reporting is considered imprecise.

If the processor core cannot be halted, the special debug interrupt can be used. With this configuration, TRC bits of the TDR equal to $2, the breakpoint trigger is converted into a debug interrupt to the processor. This interrupt 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 immediately (before the execution of the targeted instruction). This is possible because the PC breakpoint comparison is enabled at the same time the interrupt sampling occurs. For the address and data breakpoints, the reporting is considered imprecise because several additional instructions may be executed after the triggering address or data is seen.

Once the debug interrupt is recognized, the processor aborts execution and initiates exception processing. At the initiation of the exception processing, the core enters emulator mode. 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).

Execution continues at the instruction address contained in this exception vector. All interrupts are ignored while in emulator mode. Users can program the debug-interrupt handler to perform the necessary context saves using the supervisor instruction set. As an example, this handler may save the state of all the program-visible registers as well as the current context into a reserved memory area.

Once the required operations are completed, the return-from-exception (RTE) instruction is executed and the processor exits emulator mode. Once the debug interrupt handler has completed its execution, the external development system can then access the reserved memory locations using the BDM commands to read memory.

Prior to the Rev. A implementation, if a hardware breakpoint (For example, a PC trigger) is left unmodified by the debug interrupt service routine, another debug interrupt is generated after the RTE instruction completes execution. In the Rev. A design, the hardware has been modified to inhibit the generation of another debug interrupt during the first instruction after the RTE exits emulator mode. This behavior is consistent with the existing logic involving trace mode, where the execution of the first instruction occurs before another trace exception is generated. This Rev. A enhancement disables all hardware breakpoints until the first instruction after the RTE has completed execution, regardless of the programmed trigger response.

20.4.1.1 Emulator Mode

Emulator mode is used to facilitate non-intrusive emulator functionality. This mode can be entered in three different ways:

- The EMU bit in the CSR may be programmed to force the ColdFire processor to begin execution in emulator mode. This bit is only examined when $\overline{\text{RSTI}}$ is negated and the processor begins reset exception processing. It may be set while the processor is halted before the reset exception processing begins. Refer to Section 20.3.1, "CPU Halt."
- A debug interrupt always enters emulation mode when the debug interrupt exception processing begins.
- The TCR bit in the CSR may be programmed to force the processor into emulation mode when trace exception processing begins.

During emulation mode, the ColdFire processor exhibits the following properties:

- All interrupts are ignored, including level seven.
- If the MAP bit of the CSR is set, all memory accesses are forced into a specially mapped address space signalled by TT = $2, TM = $5 or $6. This includes the stack frame writes and the vector fetch for the exception which forced entry into this mode.
- If the MAP bit in the CSR is set, all caching of memory accesses is disabled. Additionally, the SRAM module is disabled while in this mode.

The return-from-exception (RTE) instruction exits emulation mode. The processor status output port provides a unique encoding for emulator mode entry ($D) and exit ($7).

20.4.1.2 Debug Module Hardware

20.4.1.2.1 Reuse of Debug Module Hardware (Rev. A)

The debug module implementation provides a common hardware structure for both BDM and breakpoint functionality. Several structures are used for both BDM and breakpoint purposes. Table 20-18 identifies the shared hardware structures.

Table 20-18. 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

The shared use of these hardware structures means the loading of the register to perform any specified function 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 the breakpoint. If a data breakpoint is configured, a BDM write command overwrites the breakpoint contents.

20.5 Debug Module Memory Map and Register Definitions

In addition to the existing BDM commands that provide access to the processor's registers and the memory subsystem, the debug module contains nine registers to support the required functionality. All of these registers are treated as 32-bit quantities, regardless of the actual number of bits in the implementation. The registers, known as the debug control registers, are accessed through the BDM port using two new BDM commands: WDMREG and RDMREG. These commands contain a 4-bit field, DRc, which specifies the particular register being accessed.

These registers are also accessible from the processor's supervisor programming model through the execution of the WDEBUG instruction. Thus, the breakpoint hardware within the debug module may be accessed by the external development system using the serial interface, or by the operating system running on the processor core. It is the responsibility of the software to guarantee that all accesses to these resources are serialized and logically consistent. The hardware provides a locking mechanism in the CSR to allow the external development system to disable any attempted writes by the processor to the breakpoint registers (setting IPW = 1). The BDM commands must not be issued if the ColdFire processor is accessing the debug module registers using the WDEBUG instruction.

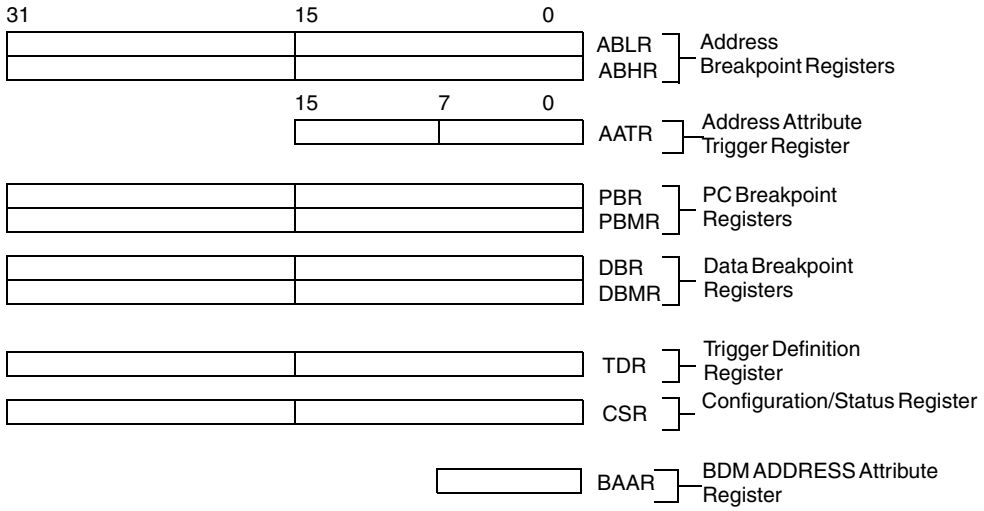


Figure 20-28. Debug Programming Mode

20.5.1 Address Breakpoint Registers

The address breakpoint registers (ABLR and ABHR) define a region in the operand address space of the processor that can be used as part of the trigger. The full 32-bits of the ABLR and ABHR values are compared with the address for all transfers on the processor's high-speed local bus. The trigger definition register (TDR) determines if the trigger is the inclusive range bound by ABLR and ABHR, all addresses outside this range, or the address in ABLR only. The ABHR is accessible in supervisor mode as debug control register $C using the WDEBUG instruction and through the BDM port using the RDMREG and WDMREG commands. The ABLR is accessible in supervisor mode as debug control register $D using the WDEBUG instruction and through the BDM port using the WDMREG commands. The ABHR is overwritten by the BDM hardware when accessing memory as described in Section 20.4.1.2, "Debug Module Hardware."

Access: User write only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	ADDRESS[31:0]															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	ADDRESS[31:0]															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 20-29. Address Breakpoint Low Register (ABLR)

ADDRESS[31:0]–Low Address

This field contains the 32-bit address which marks the lower bound of the address breakpoint range. Additionally, if a breakpoint on a specific address is required, the value is programmed into the ABLR.

Access: User write only

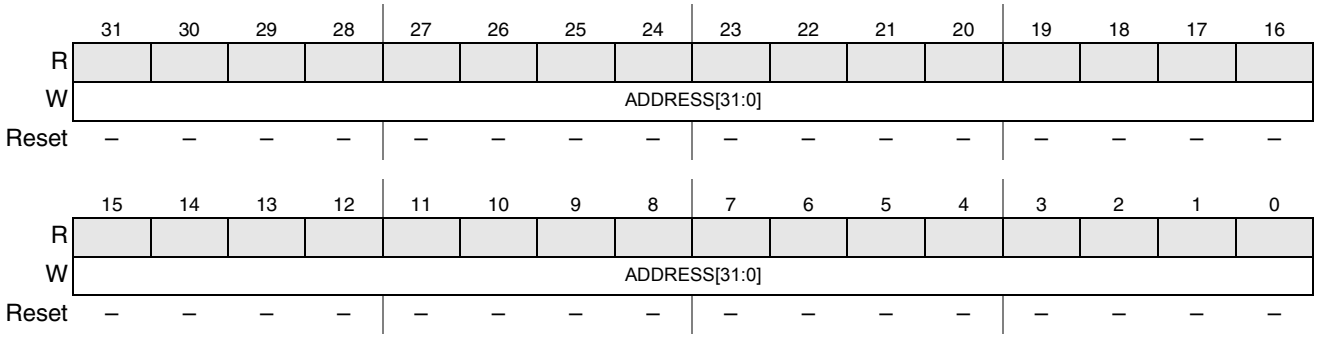


Figure 20-30. Address Breakpoint High Register (ABHR)

ADDRESS[31:0]–High Address

This field contains the 32-bit address which marks the upper bound of the address breakpoint range.

20.5.2 Address Attribute Trigger Register

The AATR defines the address attributes and a mask to be matched in the trigger. The AATR value is compared with the address attribute signals from the processor's local high-speed bus, as defined by the setting of the TDR. The AATR is accessible in supervisor mode as debug control register $6 using the WDEBUG instruction and through the BDM port using the WDMREG command. The lower five bits of the AATR are also used for BDM command definition to define the address space for memory references as described in Section 20.4.1.2, "Debug Module Hardware."

Access: User write only

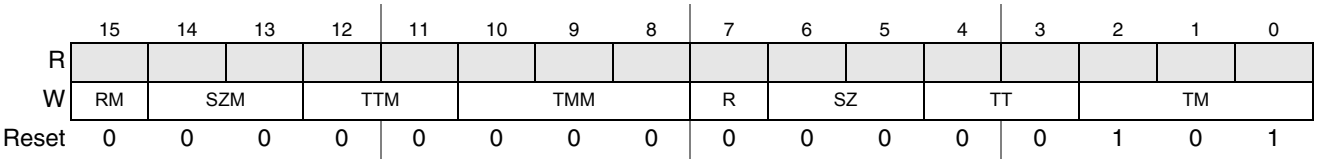


Figure 20-31. Address Attribute Trigger Register (AATR)

Table 20-19. Address Attribute Trigger Register Field Descriptions

Field	Description
15 RM	The Read/Write Mask field corresponds to the R-field. Setting this bit causes R to be ignored in address comparisons.
14–13 SZM	The Size Mask field corresponds to the SZ field. Setting a bit in this field causes the corresponding bit in SZ to be ignored in address comparisons.
12–11 TTM	The Transfer Type Mask field corresponds to the TT field. Setting a bit in this field causes the corresponding bit in TT to be ignored in address comparisons.
10–8 TMM	The Transfer Modifier Mask field corresponds to the TM field. Setting a bit in this field causes the corresponding bit in TM to be ignored in address comparisons.
7 R	The Read/Write field is compared with the R/W signal of the processor's local bus.

Table 20-19. Address Attribute Trigger Register Field Descriptions

Field	Description
6–5 SZ	The Size field is compared to the size signals of the processor's local bus. These signals indicate the data size for the bus transfer. 00 Longword 01 Byte 10 Word 11 Reserved
4–3 TT	The transfer type field is compared with the transfer type signals of the processor's local bus. These signals indicate the transfer type for the bus transfer. These signals are always encoded as if the ColdFire is in the ColdFire IACK mode. 00 Normal Processor Access 01 Reserved 10 Emulator Mode Access 11 Acknowledge/CPU Space Access These bits also define the TT encoding for BDM memory commands. In this case, the 01 encoding generates an alternate master access (for backward compatibility).
2–0 TM	The transfer modifier field is compared with the transfer modifier signals of the processor's local bus. The signals provide supplemental information for each transfer type. The encoding for normal processor transfers (TT = 0) is: 000 Explicit Cache Line Push 001 User Data Access 010 User Code Access 011 Reserved 100 Reserved 101 Supervisor Data Access 110 Supervisor Code Access 111 Reserved The encoding for emulator mode transfers (TT = 10) is: 0xx Reserved 100 Reserved 101 Emulator Mode Data Access 110 Emulator Mode Code Access 111 Reserved The encoding for acknowledge/CPU space transfers (TT = 11) is: 000 CPU Space Access 001 Interrupt Acknowledge Level 1 010 Interrupt Acknowledge Level 2 011 Interrupt Acknowledge Level 3 100 Interrupt Acknowledge Level 4 101 Interrupt Acknowledge Level 5 110 Interrupt Acknowledge Level 6 111 Interrupt Acknowledge Level 7 These bits also define the TM encoding for BDM memory commands (For backward compatibility).

20.5.3 Program Counter Breakpoint Register (PBR, PBMR)

The PC breakpoint registers (PBR and PBMR) define a region in the code address space of the processor that can be used as part of the trigger. The PBR value is masked by the PBMR value, allowing only those bits in PBR that have a corresponding zero in PBMR to be compared with the processor's program counter register, as defined in the TDR. The PBR is accessible in supervisor mode as debug control register $8 using the WDEBUG instruction and through the BDM port using the RDMREG and WDMREG

commands. The PBMR is accessible in supervisor mode as debug control register $9 using the WDEBUG instruction and through the BDM port using the WDMREG command.

Access: User write only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	ADDRESS[31:0]															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	ADDRESS[31:0]															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 20-32. Program Counter Breakpoint Register (PBR)

ADDRESS[31:0]–PC Breakpoint Address

This field contains the 32-bit address to be compared with the PC as a breakpoint trigger.

Access: User write only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	MASK [31:0]															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	MASK [31:0]]															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 20-33. Program Counter Breakpoint Mask Register (PBMR)

MASK[31:0]–PC Breakpoint Mask

This field contains the 32-bit mask for the PC breakpoint trigger. A zero in a bit position causes the corresponding bit in the PBR to be compared to the appropriate bit of the PC. A one causes that bit to be ignored.

20.5.4 Data Breakpoint Registers (DBR, DBMR)

The data breakpoint registers (DBR and DBMR) define a specific data pattern that can be used as part of the trigger into debug mode.The DBR value is masked by the DBMR value, allowing only those bits in DBR that have a corresponding zero in DBMR to be compared with the data value from the processor's local bus, as defined in the TDR. The DBR is accessible in supervisor mode as debug control register $E using the WDEBUG instruction and through the BDM port using the RDMREG and WDMREG commands. The DBMR is accessible in supervisor mode as debug control register $F using the WDEBUG instruction and through the BDM port using the WDMREG command. The DBR is overwritten by the BDM hardware when accessing memory as described in Section 20.4.1.2, "Debug Module Hardware."

Access: User write only

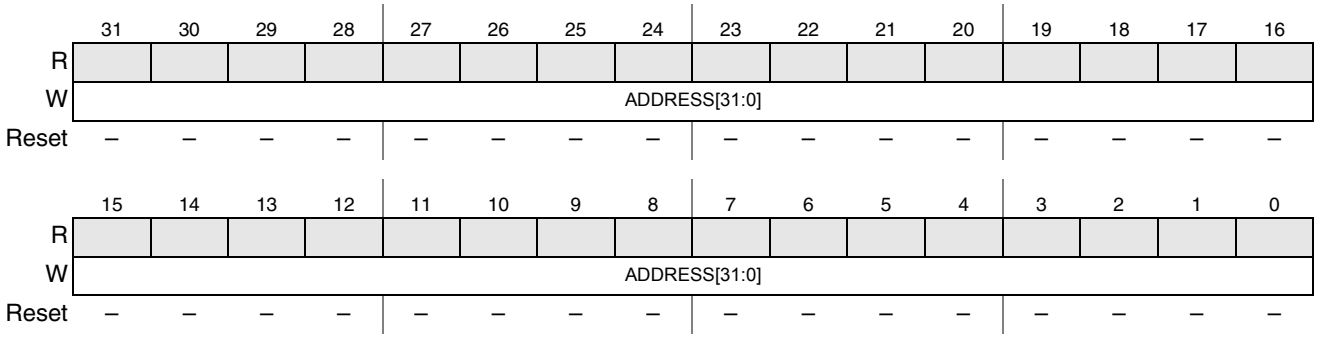


Figure 20-34. Data Breakpoint Register (DBR)

DATA[31:0]–Data Breakpoint Value

This field contains the 32-bit value to be compared with the data value from the processor's local bus as a breakpoint trigger.

Access: User write only

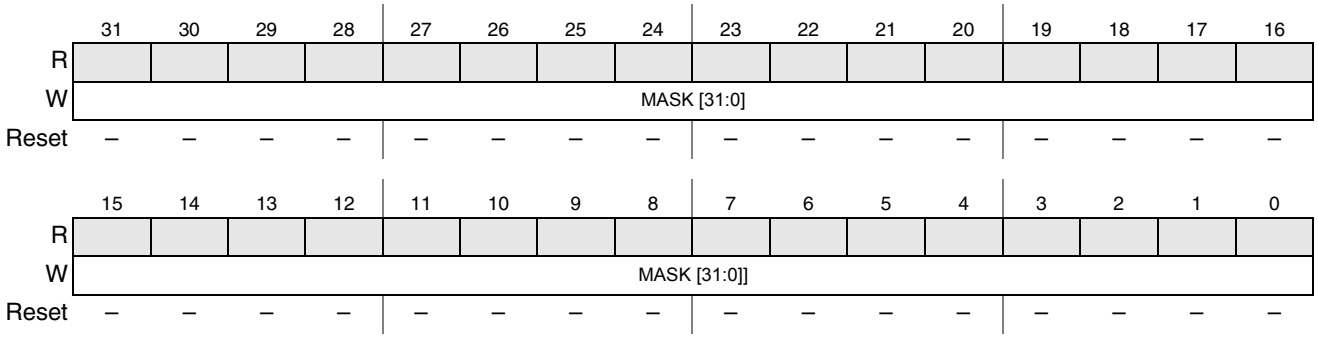


Figure 20-35. Data Breakpoint Mask Register (DBMR)

MASK[31:0]–Data Breakpoint Mask

This field contains the 32-bit mask for the data breakpoint trigger. A zero in a bit position causes the corresponding bit in the DBR to be compared to the appropriate bit of the internal data bus. A one causes that bit to be ignored.

The data breakpoint register supports both aligned and misaligned references. The relationship between the processor address, the access size, and the corresponding location within the 32-bit data bus is shown in Table 20-20.

Table 20-20. Access and Operand Data Location

Address[1:0]	Access Size	Operand Location
00	Byte	Data[31:24]
01	Byte	Data[23:16]
10	Byte	Data[15:8]
11	Byte	Data[7:0]
0x	Word	Data[31:16]
1x	Word	Data[15:0]
xx	Long	Data[31:0]

20.5.5 Trigger Definition Register (TDR)

The TDR configures the operation of the hardware breakpoint logic within the debug module and controls the actions taken under the defined conditions. The breakpoint logic may be configured as a one- or two-level trigger, where bits [31:16] of the TDR define the 2nd level trigger and bits [15:0] define the first level trigger. The TDR is accessible in supervisor mode as debug control register $7 using the WDEBUG instruction and through the BDM port using the WDMREG command.

Access: User write only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	TRC			EDLW	EDWL	EDWU	EDLL	EDLM	EDUM	EDUU	DI	EAI	EAR	EAL	EPC	PCI
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W			EBL	EDLW	EDWL	EDWU	EDLL	EDLM	EDUM	EDUU	DI	EAI	EAR	EAL	EPC	PCI
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 20-36. Trigger Definition Register (TDR)

Table 20-21. Trigger Definition Register (TDR) Field Descriptions

Field	Description
31–29 TRC	The trigger response control determines how the processor is to respond to a completed trigger condition. The trigger response is always displayed on the DDATA pins. 00 Display on DDATA only 01 Processor halt 10 Debug interrupt 11 Reserved
15 TDR	0 Level-2 trigger = PC_condition & Address_range & Data_condition 1 Level-2 trigger = PC_condition | (Address_range & Data_condition)
14 TDR	0 Level-1 trigger = PC_condition & Address_range & Data_condition 1 Level-1 trigger = PC_condition | (Address_range & Data_condition)

Table 20-21. Trigger Definition Register (TDR) Field Descriptions

Field	Description
13 EBL	If set, the Enable Breakpoint Level bit serves as the global enable for the breakpoint trigger. If cleared, all breakpoints are disabled.
28, 12 EDLW	If set, the Enable Data Breakpoint for the Data Longword bit enables the data breakpoint based on the entire processor's local data bus. The assertion of any of the ED bits enables the data breakpoint. If all bits are cleared, the data breakpoint is disabled.
27, 11 EDWL	If set, the Enable Data Breakpoint for the Lower Data Word bit enables the data breakpoint based on the low-order word of the processor's local data bus.
26, 10 EDWU	If set, the Enable Data Breakpoint for the Upper Data Word bit enables the data breakpoint trigger based on the high-order word of the processor's local data bus.
25, 9 EDLL	If set, the Enable Data Breakpoint for the Lower Lower Data Byte bit enables the data breakpoint trigger based on the low-order byte of the low-order word of the processor's local data bus.
24, 8 EDLM	If set, the Enable Data Breakpoint for the Lower Middle Data Byte bit enables the data breakpoint trigger based on the high-order byte of the low-order word of the processor's local data bus.
23, 7 EDUM	If set, the Enable Data Breakpoint for the Upper Middle Data Byte bit enables the data breakpoint trigger on the low-order byte of the high-order word of the processor's local data bus.
22, 6 EDUU	If set, the Enable Data Breakpoint for the Upper Upper Data Byte bit enables the data breakpoint trigger on the high-order byte of the high-order word of the processor's local data bus.
21, 5 DI	The Data Breakpoint Invert bit provides a mechanism to invert the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value not equal to the one programmed into the DBR.
20, 4 EAI	If set, the Enable Address Breakpoint Inverted bit enables the address breakpoint based outside the range defined by ABLR and ABHR. The assertion of any of the EA bits enables the address breakpoint. If all three bits are cleared, this breakpoint is disabled.
19, 3 EAR	If set, the Enable Address Breakpoint Range bit enables the address breakpoint based on the inclusive range defined by ABLR and ABHR.
18, 2 EAL	If set, the Enable Address Breakpoint Low bit enables the address breakpoint based on the address contained in the ABLR.
17, 1 EPC	If set, the Enable PC Breakpoint bit enables the PC breakpoint.
16, 0 PCI	If set, the PC Breakpoint Invert bit allows execution outside a given region as defined by PBR and PBMR to enable a trigger. If cleared, the PC breakpoint is defined within the region defined by PBR and PBMR.

20.5.6 Configuration/Status Register (CSR)

The CSR defines the debug configuration for the processor and memory subsystem. In addition to defining the microprocessor configuration, this register also contains status information from the breakpoint logic. The CSR is cleared during system reset. The CSR can be read and written by the external development system and written by the supervisor programming model. The CSR is accessible in supervisor mode as debug control register $0 using the WDEBUG instruction and through the BDM port using the RDMREG and WDMREG commands.

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	STATUS				FOF	TRG	HALT	BKPT	HRL					BKD	PCD	IPW
W																
Reset	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	MAP	TRC	EMU	DDC		UHE	BTB			NPL	IPI	SSM				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	–	–	–	–

Note: The CSR is a write only register from the programming model. It can be read from and written to through the BDM port.

Figure 20-37. Configuration/Status Register (CSR)

Table 20-22. Configuration/Status Register (CSR) Field Descriptions

Field	Description
31–28 STATUS	The Breakpoint Status 4-bit field provides read-only status information concerning the hardware breakpoints. This field is defined as follows: 000x No breakpoints enabled 001x Waiting for level 1 breakpoint 010x Level 1 breakpoint triggered 101x Waiting for level 2 breakpoint 110x Level 2 breakpoint triggered The breakpoint status is also output on the DDATA port when it is not busy displaying other processor data. A write to the TDR resets this field.
27 FOF	If the read-only Fault-on-Fault status bit is set, a catastrophic halt has occurred and forced entry into BDM. This bit is cleared on a read from the CSR.
26 TRG	If the read-only Hardware Breakpoint Trigger status bit is set, a hardware breakpoint has halted the processor core and forced entry into BDM. This bit is cleared by reading CSR.
25 HALT	If the read-only Processor Halt status bit is set, the processor has executed the HALT instruction and forced entry into BDM. This bit is cleared by reading the CSR.
24 $\overline{\text{BKPT}}$	If the read-only Breakpoint Assert status bit is set, the $\overline{\text{BKPT}}$ signal was asserted, forcing the processor into BDM. This bit is cleared on a read from the CSR.
23–20 HRL	This hardware revision level indicates the level of functionality implemented in the debug module. This information could be used by an emulator to identify the level of functionality supported. A zero value would indicate the initial debug functionality. For example, a value of 1 would represent Revision A while a value of 0 would represent the earlier release of Revision A.
18 BKD	The Disable the Normal $\overline{\text{BKPT}}$ Input Signal Functionality bit is used to disable the normal $\overline{\text{BKPT}}$ input signal functionality, and allow the assertion of this pin to generate a debug interrupt. If set, the assertion of the $\overline{\text{BKPT}}$ pin is treated as an edge-sensitive event. Specifically, a high-to-low edge on the $\overline{\text{BKPT}}$ pin generates a signal to the processor indicating a debug interrupt. The processor makes this interrupt request pending until the next sample point occurs. At that time, the debug interrupt exception is initiated. In the ColdFire architecture, the interrupt sample point occurs once per instruction. There is no support for any type of “nesting” of debug interrupts.
17 PCD	If set, the PSTCLK Disable bit disables the generation of the PSTCLK output signal, and forces this signal to remain quiescent.

Table 20-22. Configuration/Status Register (CSR) Field Descriptions (continued)

Field	Description
16 IPW	If set, the Inhibit Processor Writes to Debug Registers bit inhibits any processor-initiated writes to the debug module's programming model registers. This bit can only be modified by commands from the external development system.
15 MAP	If set, the Force Processor References in Emulator Mode bit forces the processor to map all references while in emulator mode to a special address space, TT = $2, TM = $5 or $6. If cleared, all emulator-mode references are mapped into supervisor code and data spaces.
14 TRC	If set, the Force Emulation Mode on Trace Exception bit forces the processor to enter emulator mode when a trace exception occurs.
13 EMU	If set, the Force Emulation Mode bit forces the processor to begin execution in emulator mode. Refer to Section 20.4.1.1, "Emulator Mode."
12–11 DDC	The 2-bit Debug Data Control field provides configuration control for capturing operand data for display on the DDATA port. The encoding is: 00 No operand data is displayed 01 Capture all M-Bus write data 10 Capture all M-Bus read data 11 Capture all M-Bus read and write data In all cases, the DDATA port displays the number of bytes defined by the operand reference size. For example, byte displays 8 bits, word displays 16 bits, and long displays 32 bits (one nibble at a time across multiple clock cycles.) Refer to Section 20.2.1.7, "Begin Data Transfer (PST = $8–$B)."
10 UHE	The User Halt Enable bit selects the CPU privilege level required to execute the HALT instruction. 0 HALT is a privileged, supervisor-only instruction 1 HALT is a non-privileged, supervisor/user instruction
9–8 BTB	The 2-bit Branch Target Bytes field defines the number of bytes of branch target address to be displayed on the DDATA outputs. The encoding is: 00 0 bytes 01 Lower two bytes of the target address 10 Lower three bytes of the target address 11 Entire four-byte target address Refer to Section 20.2.1.5, "Begin Execution of Taken Branch (PST = $5)."
6 NPL	If set, the Non-Pipelined Mode bit forces the processor core to operate in a nonpipeline mode of operation. In this mode, the processor effectively executes a single instruction at a time with no overlap. When operating in non-pipelined mode, performance is severely degraded. For the V3 design, operation in this mode essentially adds 6 cycles to the execution time of each instruction. Given that the measured Effective Cycles per Instruction for V3 is ~2 cycles/instruction, meaning performance in non-pipeline mode would be ~8 cycles/instruction, or approximately 25% compared to the pipelined performance. Regardless of the state of CSR[6], if a PC breakpoint is triggered, it is always reported before the instruction with the breakpoint is executed. The occurrence of an address and/or data breakpoint trigger is imprecise in normal pipeline operation. When operating in non-pipeline mode, these triggers are always reported before the next instruction begins execution. In this mode, the trigger reporting can be considered to be precise. As previously detailed, the occurrence of an address and/or data breakpoint should always happen before the next instruction begins execution. Therefore the occurrence of the address/data breakpoints should be guaranteed.

Table 20-22. Configuration/Status Register (CSR) Field Descriptions (continued)

Field	Description
5 IPI	If set, the Ignore Pending Interrupts bit forces the processor core to ignore any pending interrupt requests signalled while executing in single-instruction-step mode.
4 SSM	If set, the Single-Step Mode bit forces the processor core to operate in a single-instruction-step mode. While in this mode, the processor executes a single instruction and then halts. While halted, any of the BDM commands may be executed. On receipt of the GO command, the processor executes the next instruction and then halts again. This process continues until the single-instruction-step mode is disabled.

20.5.7 BDM Address Attribute Register (BAAR)

The BAAR register defines the address space for memory-referencing BDM commands. Bits [7:5] 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 the Rev. A implementation, this register is loaded any time the AATR is written. The BAR is initialized to a value of $5, setting “supervisor data” as the default address space.

Access: User write only

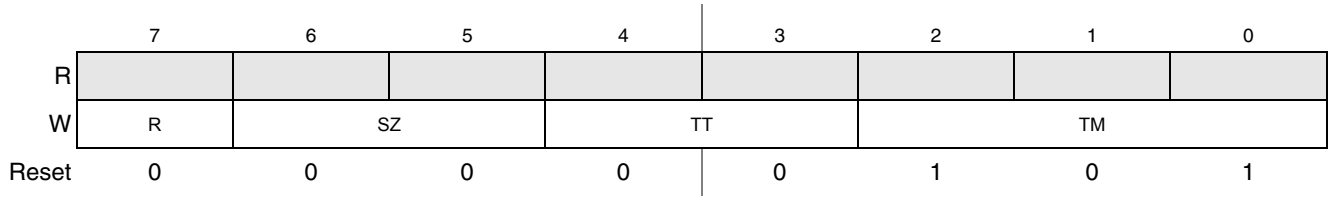


Figure 20-38. BDM Address Attribute Register (BAAR)

Table 20-23. BDM Address Attribute (BAAR) Register Field Descriptions

Field	Description
7 R	0 Write 1 Read
6–5 SZ	Size 00 Longword 01 Byte 10 Word 11 Reserved
4–3 TT	Transfer Type See the TT definition in the AATR description, Section 20.5.2, “Address Attribute Trigger Register.”
2–0 TM	Transfer Modifier See the TM definition in the AATR description, Section 20.5.2, “Address Attribute Trigger Register.”

20.5.8 Concurrent BDM and Processor Operation

The debug module supports concurrent operation of both the processor and most BDM commands. BDM commands may be executed while the processor is running, except for the operations that access processor/memory registers as follows:

- Read/Write Address and Data Registers
- Read/Write Control Registers

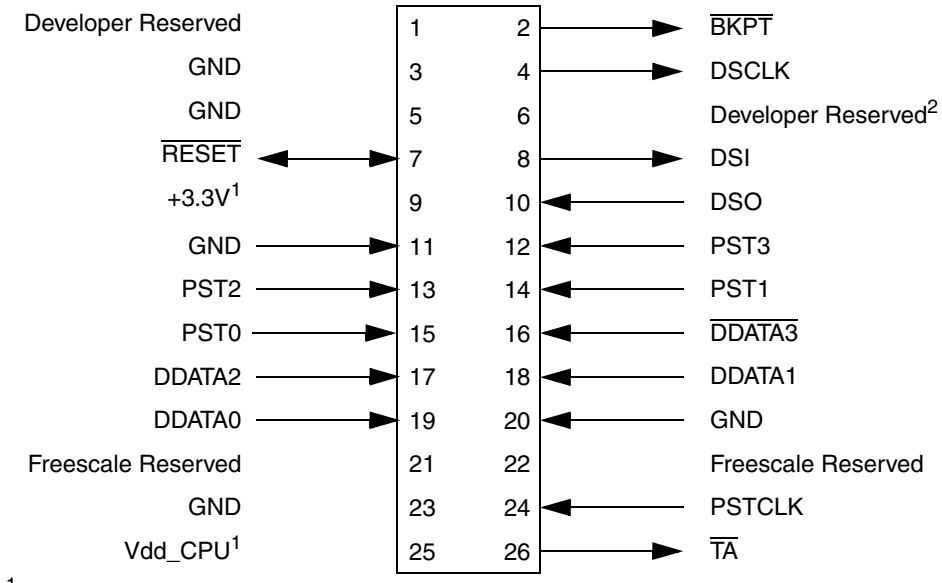
For BDM commands that access memory, the debug module requests the processor's local bus. The processor responds by stalling the instruction fetch pipeline and then waiting until all current bus activity is complete. At that time, the processor relinquishes the local bus to allow the debug module to perform the required operation. After the conclusion of the debug module bus cycle, the processor reclaims ownership of the bus.

The development system must use caution in configuring the breakpoint registers if the processor is executing. The debug module does not contain any hardware interlocks, so Freescale recommends that the TDR be disabled while the breakpoint registers are being loaded. At the conclusion of this process, the TDR can be written to define the exact trigger. This approach guarantees that no spurious breakpoint triggers occur.

Because there are no hardware interlocks in the debug unit, no BDM operations are allowed while the CPU is writing the debug's registers ($\overline{\text{BKPT}}$ and DSCLK must be inactive).

20.5.9 Freescale-Recommended BDM Pinout

The ColdFire BDM connector is a 26-pin Berg Connector arranged 2x13, shown in Figure 20-39.



1 Supplied by target

2 Pins reserved for BDM developer use. Contact developer.

Figure 20-39. Recommended BDM Connector

Chapter 21
IEEE 1149.1 Test Access Port (JTAG)

This chapter discussed the JTAG signal descriptions, TAP controller, register descriptions, and how to disable the standard operation.

The MCF5251 JTAG test architecture implementation currently supports circuit board test strategies that are based on the IEEE standard. This architecture provides access to all of the data and chip control pins from the board edge connector through the standard four-pin test access port (TAP) and the active-low JTAG reset pin, $\overline{\text{TRST}}$. The test logic uses static design and is wholly independent of the system logic, except where the JTAG is subordinate to other complimentary test modes (see Chapter 20, "Background Debug Mode (BDM) Interface," for more information). When in subordinate mode, the JTAG test logic is placed in reset and the TAP pins can be used for other purposes in accordance with the rules and restrictions set forth using a JTAG compliance-enable pin.

21.1 Features

The MCF5251 JTAG implementation can do the following:

- Perform boundary-scan operations to test circuit board electrical continuity
- Bypass the MCF5251 by reducing the shift register path to a single cell
- Sample the MCF5251 system pins during operation and transparently shift out the result
- Set the MCF5251 output drive pins to fixed logic values while reducing the shift register path to a single cell
- Protect the MCF5251 system output and input pins from backdriving and random toggling (such as during in-circuit testing) by placing all system signal pins to high- impedance state

NOTE

The IEEE Standard 1149.1 test logic cannot be considered completely benign to those planning not to use JTAG capability. Users must observe certain precautions to ensure that this logic does not interfere with system or debug operation. Refer to Section 21.7, "Disabling IEEE 1149.1A Standard Operation."

21.2 Block Diagram

Figure 21-1 is a block diagram of the MCF5251 implementation of the 1149.1A IEEE Standard. The test logic includes several test data registers, an instruction register, instruction register control decode, and a 16-state dedicated TAP controller.

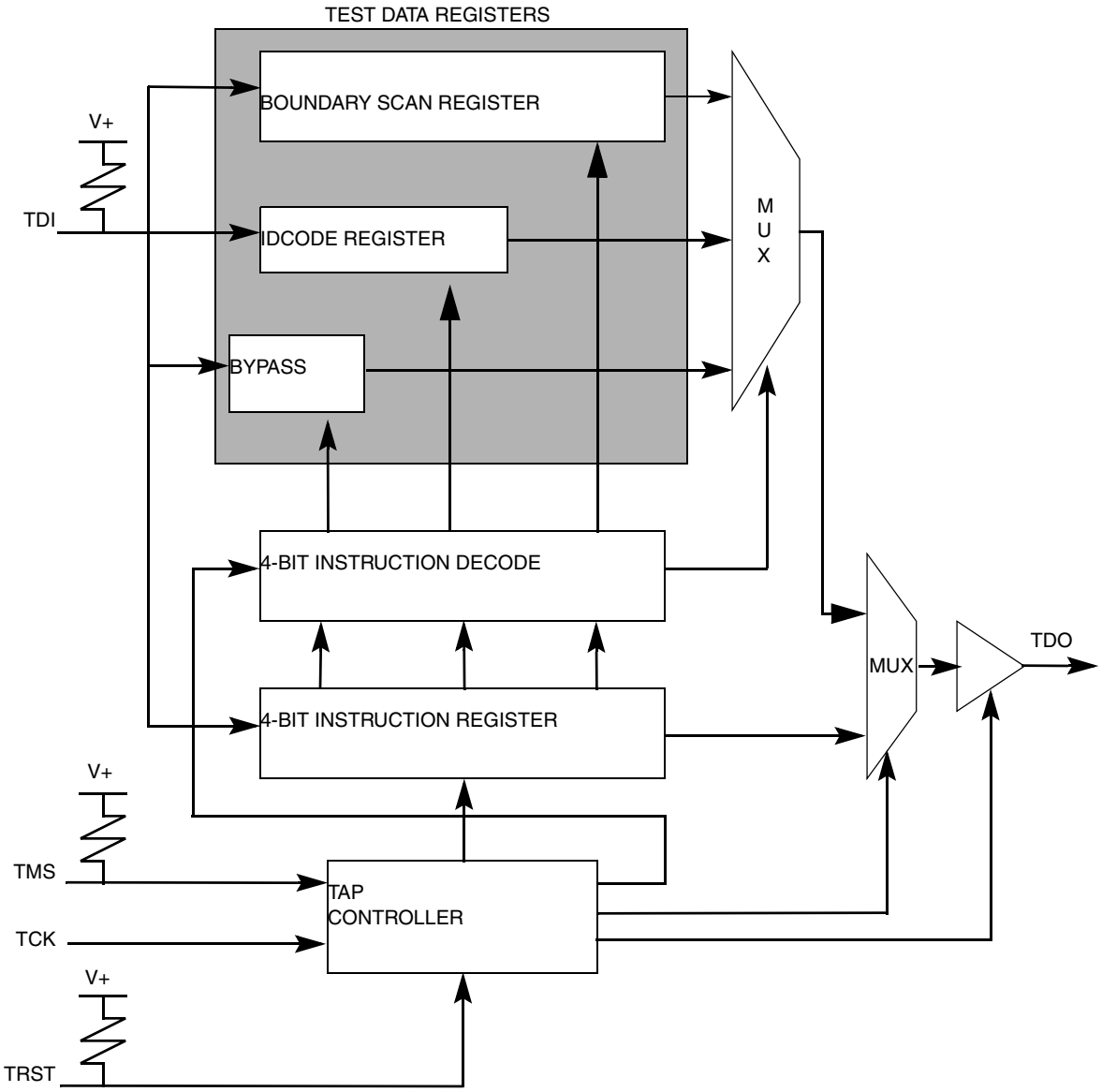


Figure 21-1. JTAG Test Logic Block Diagram

21.3 JTAG Signal Descriptions

The JTAG operation on the MCF5251 is enabled when TEST[2:0]= 000, in which case the external pin descriptions in Table 21-1 apply.Otherwise, the JTAG Test Access Port signals (TCK/TMS/TDI/TDO/TRST) are interpreted as the debug port pins.

Table 21-1. JTAG Pin Descriptions

Pin	Description
TCK	A test clock input that synchronizes test logic operations
TMS	A test mode select input with a default internal pullup resistor that is sampled on the rising edge of TCK to sequence the TAP controller

Table 21-1. JTAG Pin Descriptions

Pin	Description
TDI	A serial test data input with a default internal pullup resistor that is sampled on the rising edge of TCK.
TDO	A tri-state test data output that is actively driven only in the Shift-IR and Shift-DR controller states and only updates on the falling edge of TCK.
TRST	An active-low asynchronous reset with a default internal pullup resistor that forces the TAP controller into the test-logic-reset state.

21.3.1 Test Clock (TCK)

TCK is the dedicated JTAG test logic clock that is independent of the MCF5251 processor clock. Various JTAG operations occur on the rising or falling edge of TCK. The internal JTAG controller logic is designed such that holding TCK high or low for an indefinite period of time will not cause the JTAG test logic to lose state information. If TCK is not used, it should be tied to Vdd. There is an internal pullup connected to this pin.

21.3.2 Test Reset/Development Serial Clock ($\overline{\text{TRST}}$/DSCLK)

The TEST[2:0] signals determine the function of this dual-purpose pin. If TEST[2:0]=001, the DSCLK function is selected. If TEST[2:0]= 000, the $\overline{\text{TRST}}$ function is selected, the pin has an internal pullup and the JTAG reset is executed. For all other modes the signal is forced internally to its active value. TEST[2:0] should not be changed while $\overline{\text{RSTI}}$ is asserted.

When used as $\overline{\text{TRST}}$, this pin asynchronously resets the internal JTAG controller to the test logic reset state, causing the JTAG instruction register to choose the “idcode” command. When this occurs, all the JTAG logic is benign and will not interfere with the normal functionality of the MCF5251 processor. Although this signal is asynchronous, Freescale recommends that $\overline{\text{TRST}}$ make only a 0 to 1 (asserted to negated) transition while TMS is held at a logic 1 value. $\overline{\text{TRST}}$ has an internal pullup so that if it is not driven low its value will default to a logic level of 1. However, if $\overline{\text{TRST}}$ is not used, it can either be tied to ground or, if TCK is clocked, it can be tied to VDD. The former connection will place the JTAG controller in the test logic reset state immediately, while the later connection will cause the JTAG controller (if TMS is a logic 1) to eventually end up in the test logic reset state after 5 clocks of TCK.

This pin is also used as the development serial clock (DSCLK) for the serial interface to the debug module.The maximum frequency for the DSCLK signal is 1/2 the SYSCLK frequency.

21.3.3 Test Mode Select/ Breakpoint (TMS/$\overline{\text{BKPT}}$)

The TEST[2:0] signals determine this pin’s dual function. If TEST[2:0] =001, the $\overline{\text{BKPT}}$ function is selected. If TEST[2:0] = 000, then the TMS function is selected. TEST[2:0] should not change while $\overline{\text{RSTI}}$ is asserted. When used as TMS, this input signal provides the JTAG controller with information to determine which test operation mode should be performed. The value of TMS and current state of the internal 16-state JTAG controller state machine at the rising edge of TCK determine whether the JTAG controller holds its current state or advances to the next state. This directly controls whether JTAG data or

instruction operations occur. TMS has an internal pullup so that if it is not driven low, its value will default to a logic level of 1. However, if TMS will not be used, it should be tied to Vdd. This pin also signals a hardware breakpoint to the processor when in the debug mode.

21.3.4 Test Data Input/Development Serial Input (TDI/DSI)

This is a dual-function pin. If TEST[2:0] = 001, then DSI is selected. If TEST[2:0] = 000, then TDI is selected. When used as TDI, this input signal provides the serial data port for loading the various JTAG shift registers composed of the boundary scan register, the bypass register, and the instruction register. Shifting in of data depends on the state of the JTAG controller state machine and the instruction currently in the instruction register. This data shift occurs on the rising edge of TCK. TDI also has an internal pullup so that if it is not driven low its value will default to a logic level of 1. However, if TDI will not be used, it should be tied to VDD.

This pin also provides the single-bit communication for the debug module commands.

21.3.5 Test Data Output/Development Serial Output (TDO/DSO)

This is a dual-function pin. When TEST[2:0] = 001, then DSO is selected. When TEST[2:0] = 000, TDO is selected. When used as TDO, this output signal provides the serial data port for outputting data from the JTAG logic. Shifting out of data depends on the state of the JTAG controller state machine and the instruction currently in the instruction register. This data shift occurs on the falling edge of TCK. When TDO is not outputting test data, it is tri-stated. TDO can also be placed in tri-state mode to allow bussed or parallel connections to other devices having JTAG.

21.4 TAP Controller

The state of TMS at the rising edge of TCK determines the current state of the TAP controller. There are basically two paths that the TAP controller can follow: The first, for executing JTAG instructions; the second, for manipulating JTAG data based on the JTAG instructions. The various states of the TAP controller are shown in Figure 21-2. For more detail on each state, refer to the IEEE 1149.1A Standard JTAG document.

NOTE

From any state that the TAP controller is in, Test-Logic-Reset can be entered if TMS is held high for at least five rising edges of TCK.

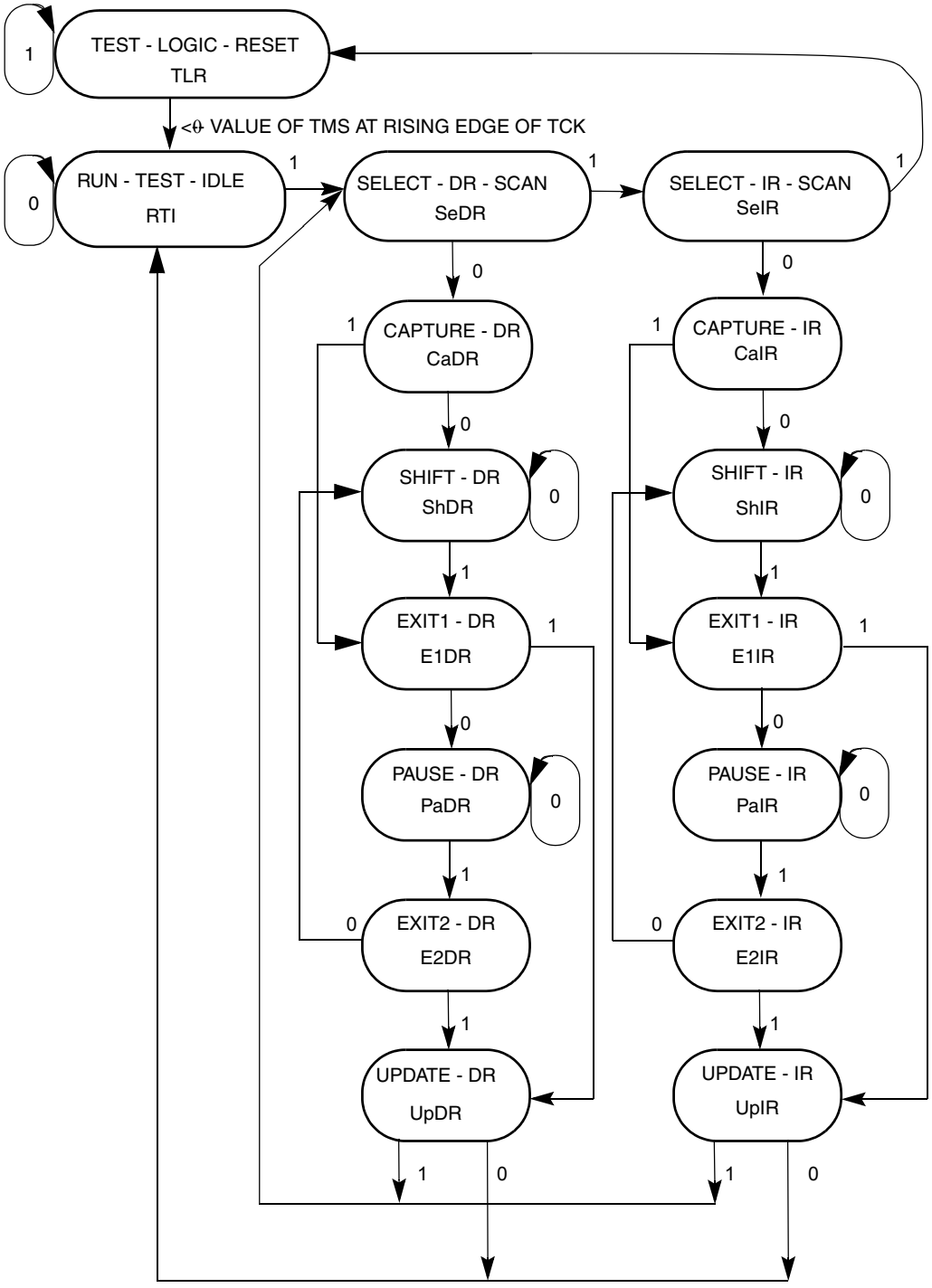


Figure 21-2. JTAG TAP Controller State Machine

21.5 JTAG Register Definitions

21.5.1 JTAG Instruction Shift Register

The MCF5251 IEEE 1149.1A Standard implementation uses a 4-bit instruction-shift register without parity. This register transfers its value to a parallel hold register and applies one of eight possible instructions on the falling edge of TCK when the TAP state machine is in the update-IR state. To load the instructions into the shift portion of the register, place the serial data on the TDI pin prior to each rising edge of TCK.

Table 21-2 lists the public, usable instructions that are supported along with their encoding.

Table 21-2. JTAG Instructions

Instruction	ABBR	Class	IR[3:0]	Instruction Summary
EXTEST	EXT	Required	0000	Select BS register while applying fixed values to output pins and asserting functional reset
IDCODE	IDC	Optional	0001	Selects IDCODE register for shift
SAMPLE/ PRELOAD	SMP	Required	0010	Selects BS register for shift, sample, and preload without disturbing functional operation
CLAMP	CMP	Optional	0011	Selects bypass while applying fixed values to output pins and asserting functional reset
HIGHZ	HI_Z	Optional	0100	Selects the bypass register while tri-stating all output pins and asserting functional reset
RINGOSC	RING	Optional	0111	User defined function for device test
ORGATE	OR	Optional	1000	User defined function for device test
BYPASS	BYP	Required	1111	Selects the bypass register for data operations

The IEEE 1149.1A Standard requires the EXTEST, SAMPLE/PRELOAD, and BYPASS instructions. IDCODE, CLAMP, HIGHZ are optional standard instructions that the MCF5251 implementation supports and are described in the IEEE Standard 1149.1A. The RINGOSC and ORGATE are user defined instructions only used for device test during manufacturing.

21.5.1.1 EXTEST Instruction

The external test instruction (EXTEST) selects the boundary-scan register. The EXTEST instruction forces all output pins and bidirectional pins configured as outputs to the preloaded fixed values (with the SAMPLE/PRELOAD instruction) and held in the boundary-scan update registers. The EXTEST instruction can also configure the direction of bidirectional pins and establish high-impedance states on some pins. The EXTEST instruction becomes active on the falling edge of TCK in the update-IR state when the data held in the instruction-shift register is equivalent to hex 0.

21.5.1.2 IDCODE

The IDCODE instruction selects the 32-bit IDcode register for connection as a shift path between the TDI pin and the TDO pin. This instruction lets users interrogate the MCF5251 to determine its version number

and other part identification data. The IDcode register has been implemented in accordance with IEEE 1149.1A so that the least significant bit of the shift register stage is set to logic 1 on the rising edge of TCK 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 set to fixed values (see Section 21.5.2, "ID Code Register") on the rising edge of TCK following entry into the capture-DR state.

The IDCODE instruction is the default value placed in the instruction register when a JTAG reset is accomplished by either asserting $\overline{\text{TRST}}$ or holding TMS high while clocking TCK through at least five rising edges and the falling edge after the fifth rising edge. A JTAG reset will cause the TAP state machine to enter the test-logic-reset state (normal operation of the TAP state machine into the test-logic-reset state will also result in placing the default value of 1 into the instruction register). The shift register portion of the instruction register is loaded with the default value of 1 when in the Capture-IR state and a rising edge of TCK occurs.

21.5.1.3 SAMPLE/PRELOAD Instruction

The SAMPLE/PRELOAD instruction provides two separate functions. First, it obtains a sample of the system data and control signals present at the MCF5251 input pins and just prior to the boundary scan cell at the output pins. This sampling occurs on the rising edge of TCK in the capture-DR state when an instruction encoding of 2 is resident in the instruction register. Users can observe this sampled data by shifting it through the boundary-scan register to the output TDO by using the shift-DR state. Both the data capture and the shift operation are transparent to system operation. Users are responsible for providing some form of external synchronization to achieve meaningful results because there is no internal synchronization between TCK and the SYSCLK.

The second function of the SAMPLE/PRELOAD instruction is to initialize the boundary scan register update cells before selecting EXTEST or CLAMP. This is achieved by ignoring the data being shifted out of the TDO pin while shifting in initialization data. The update-DR state in conjunction with the falling edge of TCK can then transfer this data to the update cells. This data will be applied to the external output pins when one of the instructions listed above is applied.

21.5.1.4 CLAMP Instruction

The CLAMP instruction selects the bypass register and asserts functional 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 registers. This instruction 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. The CLAMP instruction becomes active on the falling edge of TCK in the update-IR state when the data held in the instruction-shift register is equivalent to 3.

21.5.1.5 HIGHZ Instruction

The HIGHZ instruction anticipates the need to backdrive the output pins and protect the input pins from random toggling during circuit board testing. The HIGHZ instruction selects the bypass register, forcing all output and bidirectional pins to the high-impedance state.

The HIGHZ instruction goes active on the falling edge of TCK in the update-IR state when the data held in the instruction shift register is equivalent to 4.

21.5.1.6 BYPASS Instruction

The BYPASS instruction selects the single-bit bypass register, creating a single-bit shift register path from the TDI pin to the bypass register to the TDO pin. This instruction enhances test efficiency by reducing the overall shift path when a device other than the MCF5251 processor becomes the device under test on a board design with multiple chips on the overall IEEE1149.1A defined boundary-scan chain. The bypass register has been implemented in accordance with IEEE1149.1A so that the shift register stage is set to logic zero on the rising edge of TCK following entry into the capture-DR state. Therefore, the first bit to be shifted out after selecting the bypass register is always a logic zero (to differentiate a part that supports an IDCODE register from a part that supports only the bypass register). The BYPASS instruction goes active on the falling edge of TCK in the update-IR state when the data held in the instruction shift register is equivalent to 0xF.

21.5.2 ID Code Register

An IEEE 1149.1A compliant JTAG identification register has been included on the MCF5251. The MCF5251 JTAG instruction encoded as 1 provides for reading the JTAG IDcode register.

Address Access: User read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	VERSION NUMBER				DESIGN CENTER						DEVICE NUMBER					
W																
Reset	0	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DEVICE NUMBER				JEDECID											JTAGID
W																
Reset	0	1	1	0	0	0	0	0	0	0	0	1	1	1	0	1

Figure 21-3. ID Code Register Command

Table 21-3. ID Code Register Field Descriptions

Field	Description
31–28	The Version Number bits indicate the revision number of the MCF5251.
27–22	The Design Center bits indicate the Munich design center.
21–12	The Device Number bits indicate an MCF5251.
11–1	The JEDEC ID bits indicate the reduced JEDEC ID for Freescale (JEDEC refers to the Joint Electron Device Engineering Council. Refer to JEDEC publication 106-A and section 11 of the IEEE 1149.1A Standard for further information on this field).
0	Differentiates this register as the JTAG ID code register (as opposed to the bypass register) according to the IEEE 1149.1A Standard.

21.5.3 JTAG Boundary Scan Register

The MCF5251 model includes an IEEE 1149.1A-compliant boundary-scan register. The boundary-scan register is connected between TDI and TDO when the EXTEST or SAMPLE/PRELOAD instructions are selected. This register captures signal pin data on the input pins, forces fixed values on the output signal pins, and selects the direction and drive characteristics (a logic value or high impedance) of the bidirectional and tri-state signal pins. A detailed description of the boundary scan register bits for the MCF5251 is part of the BDSL file.

21.5.4 JTAG Bypass Register

The MCF5251 includes an IEEE 1149.1A-compliant bypass register, which creates a single bit shift register path from TDI to the bypass register to TDO when the BYPASS instruction is selected.

21.6 Restrictions

The test logic is implemented using static logic design, and TCK can be stopped in either a high or low state without loss of data. The system logic, however, operates on a different system clock which is not synchronized to TCK internally. Any mixed operation requiring the use of IEEE 1149.1A test logic in conjunction with system functional logic that uses both clocks, must have coordination and synchronization of these clocks done externally to the MCF5251.

21.7 Disabling IEEE 1149.1A Standard Operation

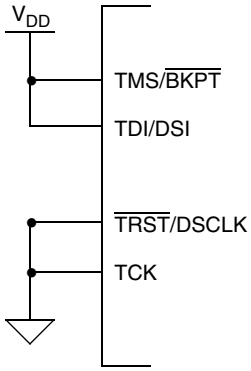
There are two ways to use the MCF5251 without the IEEE 1149.1A test logic being active:

1. Non-use of the JTAG test logic by either non-termination (disconnection) or intentional fixing of TAP logic values.
2. Intentional disabling of the JTAG test logic by setting TEST[2:0]= 001 (entering Debug mode).

There are several considerations that must be addressed if the IEEE 1149.1A logic is not going to be used once the MCF5251 is assembled onto a board.

The prime consideration is to ensure that the IEEE 1149.1A test logic remains transparent and benign to the system logic during functional operation. This requires the minimum of either connecting the $\overline{\text{TRST}}$ pin to logic 0, or connecting the TCK clock pin to a clock source that will supply five rising edges and the falling edge after the fifth rising edge, to ensure that the part enters the test-logic-reset state. The recommended solution is to connect $\overline{\text{TRST}}$ to logic 0.

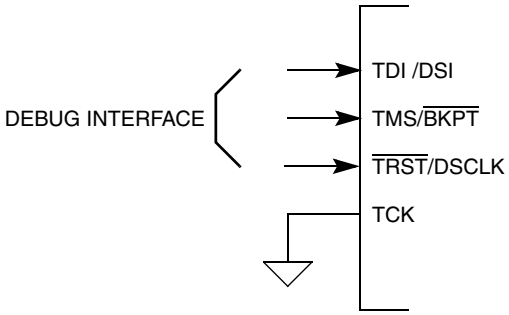
Another consideration is that the TCK pin does not have an internal pullup as is required on the TMS, TDI, and $\overline{\text{TRST}}$ pins; therefore, it should not be left unterminated to preclude mid-level input values. Figure 21-4 shows pin values recommended for disabling JTAG with the MCF5251 in JTAG mode.



NOTE: TEST[2:0] SET TO '000' ALLOWS JTAG MODE.

Figure 21-4. Disabling JTAG in JTAG Mode

A second method of using the MCF5251 without the IEEE 1149.1A logic being active is to select Debug mode by setting TEST[2:0]= 001. The IEEE 1149.1A test controller is now placed in the test-logic-reset state by the internal assertion of the $\overline{TRST}$ signal to the controller and the TAP pins function as debug mode pins. While in JTAG mode, input pins TDI/DSI, TMS/$\overline{BKPT}$, and $\overline{TRST}$/DSCLK have internal pullups enabled. Figure 21-5 shows pin values recommended for disabling JTAG with the MCF5251 in debug mode.



NOTE: TEST[2:0] SET TO' 001' PROHIBITS JTAG.

Figure 21-5. Disabling JTAG in Debug Mode

21.8 Obtaining the IEEE 1149.1A Standard

The IEEE 1149 Standard JTAG specification is a copyrighted document and must be obtained directly from the IEEE:

IEEE Standards Department
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

http://standards.ieee.org

Chapter 22
USB, ATA DMA, and Clock Integration Module

This chapter includes the memory map, register descriptions and functional description of the ATA DMA integration module.

22.1 Introduction

This chapter includes registers that are used to configure the various modules that are new to the MCF5251 family. It also includes a shared 16 kB local memory for the ATA and USB modules.

22.2 Memory Map and Register Definitions

Table 22-1 provides the memory map and register definitions for the ATA DMA integration module.

Table 22-1. Real Time Clock Memory Map

MBAR2 Offset	Register	Access	Reset Value	Section/Page
0x500	Miscellaneous Configuration Register (MISCCR)	R/W	Undefined	22.2.1/22-1
0x504	ATA DMA Source and Destination Address Register (ATA_DADDR)	R/W	0x0000_0000	22.2.2/22-3
0x508	ATA DMA Count Register (ATA_DCOUNT)	R/W	0x0000_0000	22.2.3/22-3
0x50C	RTC Time Register (RTC_TIME)	R	Undefined	26.3.2/26-2
0x510	USB/FlexCAN Clock Gating (USBCANCLK)	R/W	0x0000_0000	22.2.5/22-4

22.2.1 Miscellaneous Configuration Register (MISCCR)

Offset: MBAR2 0x500 (MISCCR) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
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			URIP	0	URIE			RTCPL	RTC CLR	DMA END	CPU END	ADIP		ADIE	ADTD	ADTA
W				URIC									ADIC			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-1. Miscellaneous Configuration Register (MISCCR)

Table 22-2. Miscellaneous Configuration Register (MISCCR) Field Descriptions

Field	Description
31–15	Reserved, must be cleared.
14	Reserved, this bit must always be set to 1.
13 URIP	USB resume interrupt pending 0 USB resume interrupt not pending 1 USB resume interrupt pending
12 URIC	USB resume interrupt clear. Always reads 0 0 no effect 1 clear USB resume interrupt
11 URIE	USB resume interrupt enable 0 Pending USB resume interrupt not sent to CPU 1 Pending USB resume interrupt forward to CPU
10	Reserved, should always be set to 0.
9	Reserved, should always be set to 0.
8 RTCPL	Real-time clock power loss. To clear this bit, set the RTCCLR bit and then clear the RTCCLR bit. 0 RTC maintaining time OK 1 Power lost to RTC
7 RTCCLR	Real-time clock power loss clear. 0 No action 1 Clears the RTCPL bit
6 DMAEND	Endianess of DMA access to ATA 0 Little endian 1 Big endian
5 CPUEND	Endianess of CPU access to ATA 0 Little endian 1 Big endian
4 ADIP	ATA DMA interrupt pending. 0 ATA DMA interrupt not pending 1 ATA DMA interrupt pending
3 ADIC	ATA DMA interrupt clear. This bit always reads 0, and clearing this bit has no affect. Write: 0 No affect 1 Clears ATA DMA interrupt
2 ADIE	ATA DMA interrupt enable. 0 ATA DMA interrupt disabled 1 ATA DMA interrupt will interrupt CPU
1 ADTD	ATA DMA channel transfer direction. 0 RAM to ATA 1 ATA to RAM
0 ADTA	ATA DMA transfer active. 1 ATA DMA armed. Will transfer data when needed as long as atadma_count > 0. 0 ATA DMA blocked. WIll not transfer data.

22.2.2 ATA DMA Address Register (ATA_DADDR)

Offset: MBAR2 0x504 (ATA_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	RAMADDR																ATAADDR															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-2. ATA DMA Address Register (ATA_DADDR)

Table 22-3. ATA DMA Address Register (ATA_DADDR) Field Descriptions

Field	Description
31–18 RAMADDR	DMA address on the RAM side of the bus.
17–16	Reserved, should be cleared.
15–2 ATAADDR	DMA address on the ATA side of the bus.
1–0	Reserved, should be cleared.

22.2.3 ATA DMA Count Register (ATA_DCOUNT)

Offset: MBAR2 0x508 (ATA_DCOUNT) Access: User read/write

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

Figure 22-3. ATA DMA Count Register (ATA_DCOUNT)

Table 22-4. ATA DMA Count Register (ATA_DCOUNT) Field Descriptions

Field	Description
31–16	Reserved, should be cleared.
15–0 COUNT	ATA DMA word count. Indicates amount of words to transfer on the ATA DMA bus.

22.2.4 RTC Time Register (RTC_TIME)

See Chapter 26, "Real-Time Clock" for a detailed description of this register.

22.2.5 USB/FlexCAN Clock Register (USBCANCLK)

Offset: MBAR2 0x510 (USBCANCLK) Access: User read/write

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

Figure 22-4. USB/FlexCAN Clock Register (USBCANCLK)

Table 22-5. USB/FlexCAN Clock Register (USBCANCLK) Field Descriptions

Field	Description
31–2	Reserved, should be cleared.
1 CANCLK	FlexCAN clock gating. 0 FlexCAN clock disabled. The FlexCAN modules are placed in sleep mode and its interface is not available. 1 FlexCAN clock enabled.
0 USBCLK	USB clock gating. 0 USB clock disabled. When disabled, the USB is placed in standby mode, and only the USB resume interrupt will work. This interrupt will signal reception of USB resume condition on the USB bus. All other functions are not available. 1 USB clock enabled.

22.3 Functional Description

22.3.1 ATA/USB Cache Memory

A 16 Kbytes local memory is put on a local bus, available as fast, local, memory to the ATA and USB modules. A block diagram of how the devices are tied to this local bus is given in Figure 22-5.

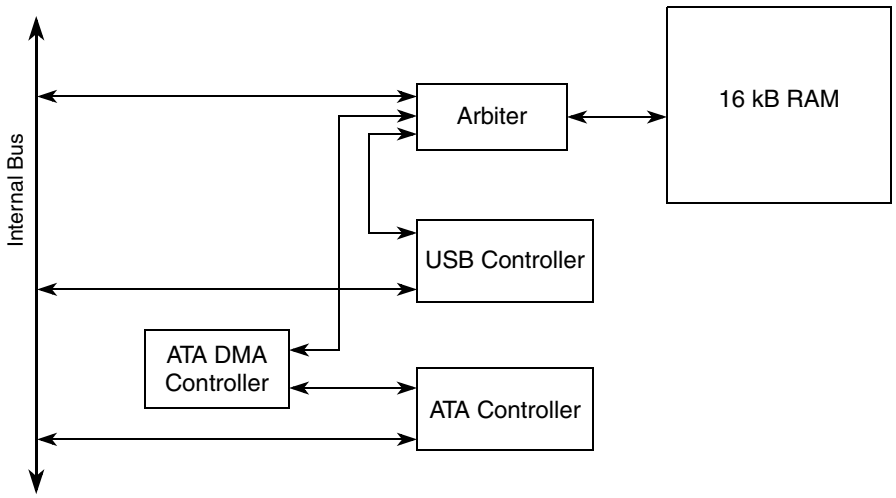


Figure 22-5. ATA/USB Shared RAM Block Diagram

The 16 kbyte SRAM is connected via an arbiter to 3 busses:

- It can be read and written directly by the USB. All transfers on the USB controller master bus are done to the 16 kB cache memory via the arbiter. USB always has priority to this memory.
- It can be read and written directly by the ColdFire CF2 core and by the ColdFire DMA engine. Note here the peripheral is connected to a slow bus, and the performance is somewhat limited.
- It can be read and written by the ATA DMA controller. This DMA controller can transfer data between the cache RAM and the ATA interface.

The ATA controller has 2 busses, one connected to the internal bus, the second connected to the local ATA DMA engine. The USB and ATA DMA controller are connected to the internal bus. For both, this connection is used to access the module registers. All USB data transfers initiated by the USB will be to and from the 16 kbytes cache memory. The USB does not see any other memory on the device.

22.3.1.1 Endianness Issues

The USB controller is little-endian, while the CPU is big-endian. The ATA controller endianness is programmable; It can operate in both little-endian and big-endian modes. To resolve the issue, the cache memory is visible from the CPU in straight-endianess and in swapped-endianness mode.

Table 22-6. USB/ATA RAM Memory Map

MBAR2 Offset	Endian
0x2_0000 ... 0x2_3FFF	Straight, read/write what's in RAM read data[31:0] = ram[31:0] write data[31:0] = in[31:0]
0x3_0000 ... 0x3_3FFF	Swapped. Read and write data swapped: read data[31:0] = {ram[7:0],ram[15:8], ram[23:16], ram[31:24]} write data[31:0] = {in[7:0], in[15:8], in[23:16], in[31:24]}

22.3.1.2 DMA Transfer between ATA and Cache RAM

The DMA of the ATA block can be used to transfer data between the ATA module and the cache RAM. In order to transfer data between these two blocks, proceed as follows:

1. Make sure no unwanted transfer will start when configuring the DMA by clearing the MISCCR[ATDA] bit.
2. Program bit MISCCR[DMAEND] to correctly reflect the endianness on the DMA you need.
3. Program ATA_DADDR[ATAADDR] to contain the address in the ATA module wherefrom or whereto the data needs to be transferred. During the transfer, this address is kept constant. It will not autoincrement or autodecrement.
   The address is offset-0, so the value that needs to be programmed here is the result of the following equation: [(ColdFire ATA register address) - (MBAR2 + 0x800)].
4. Program ATA_DADDR[RAMADDR] to contain the address in the cache RAM wherefrom or whereto the data needs to be transferred. During the transfer, this address is autoincremented.
   The address is offset-0, so the value that needs to be programmed here is the result of the following equation: [(ColdFire cache RAM address) - (MBAR2 + 0x20000)].

5. Program the number of 16-bit words that need be transferred in the ATA_DCOUNT register.
6. Program the direction of the transfer in dma_write_to_ram control bit. Set this bit for ATA to RAM transfers, clear it for RAM to ATA transfers.
7. Enable the DMA interrupt if wanted.
8. Allow the transfer to start by setting the MISCCR[ATDA] bit.
9. Start the ATA transfer. ATA will now communicate with DMA, and the ATA data automatically will end up in the memory, or data from memory is read for transfer to the ATA.
10. The DMA will end the transfer when the ATA_DCOUNT reaches zero. On DMA complete, a DMA interrupt is requested.

Chapter 23
Advanced Technology Attachment Controller (ATA)

This chapter discusses the modes of operation, signal descriptions, memory map, register descriptions, timing parameters, and functional description of the ATA interface.

The ATA interface of the MCF5251 is an AT attachment host interface. Its main use is to interface with hard disc drives and optical disc drives. It interfaces with the ATA device over a number of ATA signals.

23.1 Features

The ATA interface includes the following features:

- Programmable timing on the ATA bus. Works with wide range of bus frequencies.
- Compliant with ATA-6 specification
  - — Supports PIO modes 0, 1, 2, 3, and 4
  - — Supports multiword DMA modes 0, 1, and 2
  - — Supports ultra DMA modes 0, 1, 2, and 3 with bus clock of at least 50 MHz
- 128-byte FIFO part of interface
- FIFO receive alarm, FIFO transmit alarm to DMA unit
- Zero-wait cycles transfer between DMA bus and FIFO allows fast FIFO reading/writing

23.2 Block Diagram

Figure 23-1 illustrates the block diagram of the ATA interface.

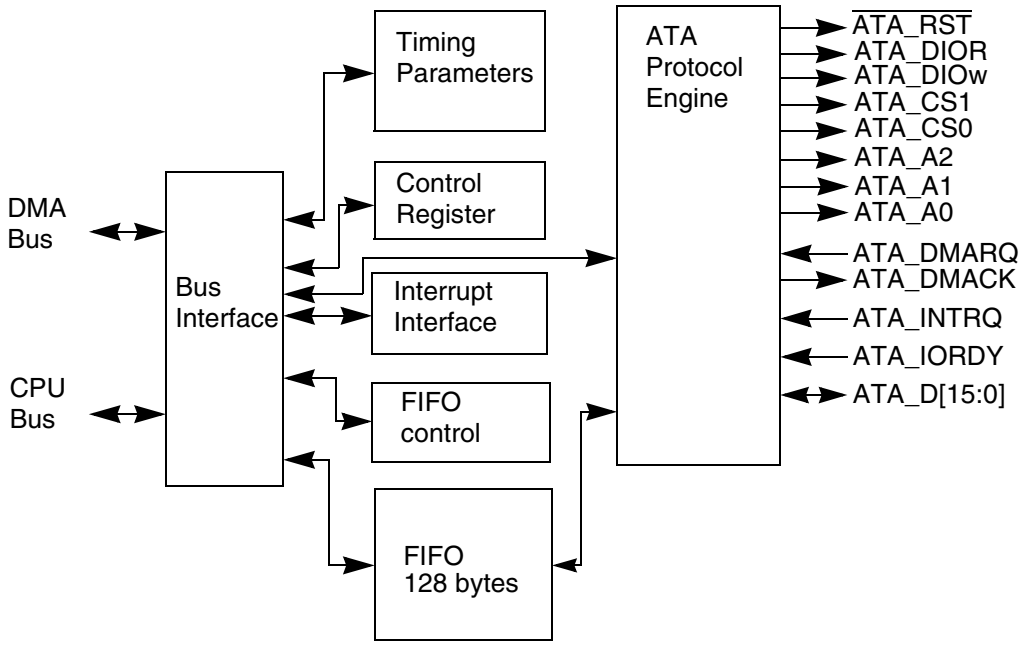


Figure 23-1. ATA Interface Block Diagram

23.3 Overview

The ATA block is a AT attachment host interface. Its main use is to interface with IDE hard disc drives and ATAPI optical disc drives. It interfaces with the ATA device over a number of ATA signals.

The ATA interface is compliant to the ATA-6 standard and supports the following protocols:

- PIO mode 0, 1, 2, 3, and 4
- multiword DMA mode 0, 1, and 2
- Ultra DMA modes 0, 1, 2, 3, and 4 with bus clock of 50 MHz or higher

The ATA interface has 2 busses connected:

- One CPU bus for communication with the host processor
- One DMA bus for communication with the host DMA unit

All internal registers are visible from both busses, allowing smart DMA access to program the interface.

Before accessing the ATA bus, the host must program the timing parameters to be used on the ATA bus. The timing parameters control the timing on the ATA bus. Most timing parameters are programmable as a number of clock cycles (1 to 255). Some are implied.

After programming the timing parameters, there are two protocols that can be active at the same time on the ATA bus:

- First protocol. This protocol is a PIO mode access that can be performed at any time by the host CPU or the host smart DMA to the ATA bus. During PIO mode access, the incoming IP bus cycle is translated to an ATA bus cycle by the ATA protocol engine. The IP bus cycle is stalled until completion of the ATA bus cycle on read, or until putting the write data on the ATA bus on write.

The PIO mode is a slow protocol, mainly intended to program the ATA disc drive, but also possible to use to transfer data to/from the disc drive. During PIO mode, the FIFO is not active.

- Second protocol. This protocol is the DMA mode access. DMA mode is started by the ATA interface after receiving a DMA request from the drive, and only if the ATA interface has been programmed to accept the DMA request. In DMA mode, either multiword DMA or ultra DMA protocol is used on the ATA bus. Once started, data transfer is organized between the ATA bus and the FIFO. Data transfer will pause to prevent FIFO overflow / FIFO underflow. Data transfer will resume when there is again space in the FIFO, or when the FIFO has been refilled. During DMA transfer, there is no direct interface between the ATA bus and the host IP or host DMA IP bus. Instead, the transfer takes place between the ATA bus and the FIFO; the FIFO informs the host DMA unit when it needs to be refilled or emptied. In this case, it sends an ALARM flag to the host DMA. When the host DMA receives the fifo_tx_alarm, it should write some data to the FIFO. (typically 32 bytes). When the host DMA receives the fifo_rcv_alarm, it should read some data from the FIFO (typically 32 bytes). The FIFO filling level at which the alarms are produced, is programmable. For completeness, there is a third alarm associated with the host DMA operation fifo_txfer_end_alarm. This alarm signals the end of the transfer, and requests the smart DMA to take steps to complete the transfer: transfer the bytes remaining in the FIFO to the host memory, and inform the host CPU the transfer is completed.

All transfers between FIFO and host IP or DMA IP bus are zero wait states transfer, so high speed transfer between FIFO and DMA/host bus is possible.

When a PIO access is performed during a running DMA transfer, the DMA transfer will be paused, the PIO access done, and the DMA transfer will resume again.

23.3.1 Modes of Operation

The interface offers two operation modes that can be used together:

- PIO Mode

  An access to the ATA bus in PIO mode occurs whenever a ATA PIO register is read or written by the host CPU or the host (smart) DMA unit. During a PIO transfer the incoming IP bus cycle is translated to an ATA PIO bus cycle by the ATA protocol engine. No buffering of data occurs, so the host CPU or host DMA cycle is stalled until the ATA bus read data is available on read, or is stalled until the IP bus data can we put on the ATA bus during write.

  PIO accesses can be done to the bus at any time, even during a running ATA DMA transfer. In this case, the DMA transfer is paused, the PIO cycle is completed, and the DMA transfer is resumed.

- DMA Mode

  In DMA mode, data is transferred between the ATA bus and the FIFO. Two different DMA protocols are supported on the ATA bus: ultra DMA mode and multiword DMA mode. Selection is made using a control register bit.

  A DMA transfer will be started when DMA mode transfer has been enabled by writing some control bit, and when the drive connected to the ATA bus pulls its DMARQ line high.

  During an ATA bus DMA transfer, data is transferred between the ATA bus and the FIFO. The transfer will pause to avoid FIFO overflow and FIFO underflow.

It is the task of the host CPU or the host smart DMA unit to read data or write data to the FIFO to keep the transfer going. Normal set-up is that the host (smart) DMA unit takes on this task. For this purpose, the *fifo_rcv_alarm* and fifo_tx_alarm signals are sent to the host DMA unit.
fifo_rcv_alarm informs the host DMA unit that there is at least 1 packet of data waiting in the FIFO to be read by the host DMA. Whenever this signal is high, the host DMA should transfer one packet of data from the FIFO to the main memory. Typical packet size is 32 bytes (8 long words), but other packet sizes can be handled too. *fifo_tx_alarm* informs the host DMA unit that there is space for at least 1 packet to be written by the host DMA. Whenever this signal is high, the host DMA should transfer one packet of data from main memory to the FIFO. Typical packet size is 32 bytes (8 long words), but other packet sizes can be handled too.

23.4 External Signal Description

See Table 23-1 for the list of signals entering and exiting this module to peripherals within the device.

Table 23-1. Signal Properties

Name	Function	Reset State	Direction
$\overline{\text{ATA\_RST}}$	ATA bus reset signal. Active low. If active, ata device is reset[1]	0	O
ATA_DIOR	ATA bus read strobe	1	O
ATA_DIOW	ATA bus write strobe	1	O
ATA_CS1	ATA bus chip select 1	1	O
ATA_CS0	ATA bus chip select 0	1	O
ATA_A2	ATA bus address line 2	0	O
ATA_A1	ATA bus address line 1	0	O
ATA_A0	ATA bus address line 0	0	O
ATA_DMARQ	ATA bus DMA request	–	I
ATA_DMACK	ATA bus DMA acknowledge	1	O
ATA_INTRQ	ATA bus interrupt request	–	I
ATA_IORDY	ATA bus iordy	–	O
ATA_D[15:0]	ATA data bus (little-endian)	HI_Z	Tri-state I/O

[1]This signal is a standard ATA bus signal. It conforms with the ATA specification.

23.4.1 Detailed Signal Descriptions

For a detailed description of the ATA bus signal, refer to the ATA-6 specification.

23.4.1.1 $\overline{\text{ATA\_RST}}$ (Out)

This signal is the ATA reset signal. When low, the ATA bus is in reset state. When high, no reset. The ATA bus is in reset whenever the appropriate bit in the control register is cleared. After system reset, the ATA bus is in reset.

23.4.1.2 ATA_DIOR (Out)

This signal correspond to ATA signal DIOR. During PIO and multiword DMA transfer, function is read strobe. During ultra DMA in burst, function is HDMARDY. During ultra DMA out burst, function is host strobe.

23.4.1.3 ATA_DIOW (Out)

This signal corresponds to ATA signal DIOW. During PIO and multiword DMA transfer, function is write strobe. During ultra DMA burst, function is STOP, signalling whenever host wants to terminate running ultra DMA transfer.

23.4.1.4 ATA_CS0, ATA_CS1, ATA_A0, ATA_A1, ATA_A2 (Out)

These signals are the address group of the ATA bus. ATA_CS0 and ATA_CS1 are the chip selects; ATA_A0, ATA_A1, and ATA_A2 are the 3 address lines. All five lines follow the same timing.

23.4.1.5 ATA_DMARQ (In)

This signal is the ATA bus device DMA request. It is pulled high by the device if it wants to transfer data using multiword DMA or ultra DMA mode.

23.4.1.6 ATA_DMACK (Out)

This signal is the ATA bus host DMA acknowledge. It is pulled low by the host when it grants the DMA request.

23.4.1.7 ATA_INTRQ (In)

This signal is the ATA bus interrupt request. It is pulled high by the device whenever it wants to interrupt the host CPU.

23.4.1.8 ATA_IORDY (In)

This signal is the ATA bus IORDY line. It has three functions:

- IORDY—active low wait during PIO cycles
- DDMARDY—active low device ready during ultra DMA out transfers
- DSTROBE—device strobe during ultra DMA in transfers

23.4.1.9 ATA_D[15:0] (In/Out/Tri-state)

This is the ATA data bus.

23.4.2 Electrical Spec on the ATA Bus, Bus Buffers

To meet electrical spec on the ATA bus, several requirements must be met. For a detailed description, refer to the ATA specification.

This electrical spec must be met for the pads used on the ATA I/Os if no bus buffers and bus transceivers are used.

Alternative is to use bus buffers. This is the only way to operate the ATA interface if 3.3 V or 5.0 V compatibility on the ATA bus is wanted, and no 3.3 V or 5.0 V tolerant pads on the device are available.

The use of bus buffers introduces delay on the bus and introduces skew between signal lines. These factors will make it difficult to operate the bus at the highest speed (UDMA-5) when bus buffers are used. If fast UDMA mode operation is needed, this may not be compatible with bus buffers.

Another area of attention is the slew rate limit imposed by the ATA specification on the ATA bus. According to this limit, any signal driven on the bus should have a slew rate between 0.4 and 1.2 V/ns with a 40 pF load. Not many vendors of bus buffers specify slew rate of the outgoing signals.

23.4.3 Timing on ATA Bus

Timing on the ATA bus is explained in this section. It is also explained how to make sure the ATA interface meets timing. Timing is explained with timing figures and also equations are provided that need to be fulfilled for the host to meet timing.

23.4.3.1 Timing Parameters

In the timing equations, some timing parameters are used. These parameters depend on the implementation of the ATA interface on silicon, the bus buffer used, the cable delay and cable skew. Refer to Table 23-2 for the list of parameters used to specify the ATA timing.

Table 23-2. Timing Parameters

Name	Meaning	Controlled by
T	Bus clock period	clock generator
ti_ds	Set-up time ATA_Dx to ATA_IORDY edge (UDMA-in only)	top level design
ti_dh	hold time ATA_IORDY edge to ATA_Dx (UDMA-in only)	top level design
tco	Propagation delay bus clock L-to-H to the following signals: ATA_CS0, ATA_CS1. ATA_CS2. ATA_A2. ATA_A1, ATA_A0, ATA_DIOR, ATA_DIOW, ATA_DMACK, ATA_Dx	top level design
tsu	Setup time ATA_Dx to bus clock L-to-H	top level design
tsui	Setup time ATA_IORDY to bus clock H-to-L	top level design
thi	Hold time ATA_IORDY to bus clock H to L	top level design
tskew1	Max difference in propagation delay bus clock L-to-H to any of following signals: ATA_CS0, ATA_CS1. ATA_CS2. ATA_A2. ATA_A1, ATA_A0, ATA_DIOR, ATA_DIOW, ATA_DMACK, ATA_Dx (write)	top level design
tskew2	Max difference in buffer propagation delay for any of following signals: ATA_CS0, ATA_CS1. ATA_CS2. ATA_A2. ATA_A1, ATA_A0, ATA_DIOR, ATA_DIOW, ATA_DMACK, ATA_Dx (write)	transceiver
tskew3	Max difference in buffer propagation delay for any of following signals: ATA_IORDY, ATA_Dx (read)	transceiver
tbuf	Max buffer propagation delay	transceiver
tcable1	Cable propagation delay for ATA_Dx	cable
tcable2	Cable propagation delay for control signals: ATA_DIOR, ATA_DIOW, ATA_IORDY, ATA_DMACK	cable

Table 23-2. Timing Parameters (continued)

Name	Meaning	Controlled by
tskew4	Max difference in cable propagation delay between ATA_IORDY and ATA_Dx (read)	cable
tskew5	Max difference in cable propagation delay between (ATA_DIOR, ATA_DIOW, ATA_DMACK) and (ATA_CS0, ATA_CS1. ATA_A2. ATA_A1, ATA_A0, ATA_Dx (write))	cable
tskew6	Max difference in cable propagation delay without accounting for ground bounce	cable

23.4.3.2 PIO Mode Timing

A timing diagram for PIO read mode is given in Figure 23-2.

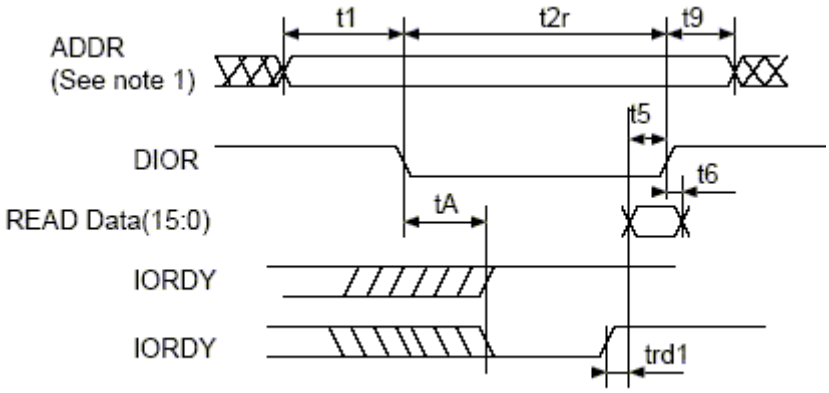


Figure 23-2. PIO Read Mode Timing

To fulfill read mode timing, the different timing parameters given in Table 23-3 must be observed.

Table 23-3. Timing Parameters PIO Read

ATA Parameter	PIO Read Mode Timing Parameter[1]	Value	How to Meet?
t1	t1	t1(min) = time_1 * T - (tskew1 + tskew2 + tskew5)	time_1
t2	t2r	t2(min) = time_2r * T - (tskew1 + tskew2 + tskew5)	time_2r
t9	t9	t9(min) = time_9 * T - (tskew1 + tskew2 + tskew6)	time_9
t5	t5	t5(min) = tco + tsu + tbuf + tbuf + tcable1 + tcable2	if not met, increase time_2
t6	t6	0	
tA	tA	tA(min) = (1.5 + time_ax) * T - (tco + tsui + tcable2 + tcable2 + 2*tbuf)	time_ax
trd	trd1	trd1(max) = (-trd) + (tskew3 + tskew4) trd1(min) = (time_pio_rdx - 0.5)*T - (tsu + thi) (time_pio_rdx - 0.5) * T > tsu + thi + tskew3 + tskew4	time_pio_rdx
t0	–	t0(min) = (time_1 + time_2 + time_9) * T	time_1, time_2r, time_9

[1] See Figure 23-2.

In PIO write mode, timing waveforms are somewhat different. A timing diagram is given in Figure 23-3.

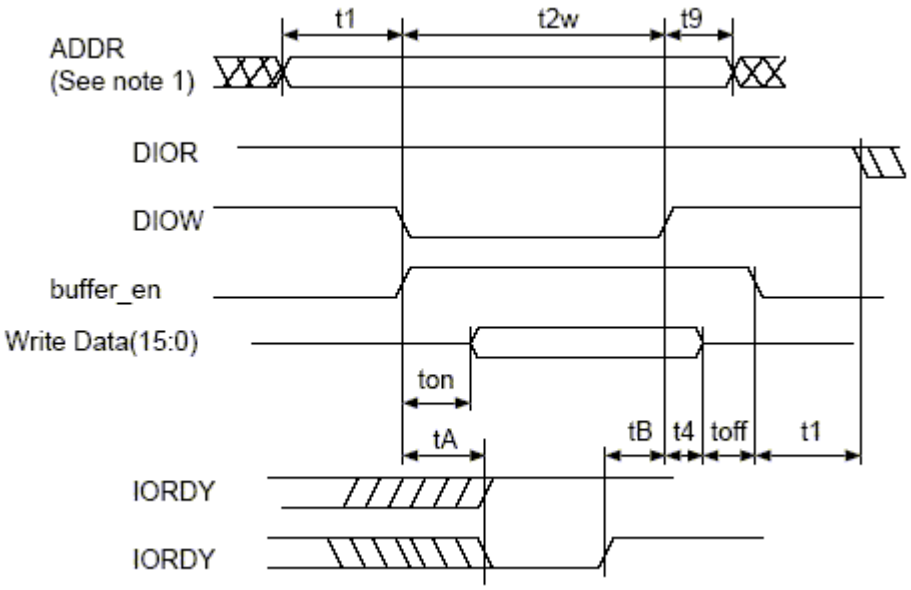


Figure 23-3. PIO Write Mode Timing

To fulfill this timing, several parameters need to be observed. See Table 23-4 for details on PIO write mode timing.

Table 23-4. Timing Parameters PIO Write

ATA Parameter	PIO Write Mode Timing Parameter[1]	Value	How to meet?
t1	t1	t1(min) = time_1 * T - (tskew1 + tskew2 + tskew5)	time_1
t2	t2w	t2(min) = time_2w * T - (tskew1 + tskew2 + tskew5)	time_2w
t9	t9	t9(min) = time_9 * T - (tskew1 + tskew2 + tskew6)	time_9
t3	–	t3(min) = (time_2w - time_on)* T - (tskew1 + tskew2 +tskew5)	if not met, increase time_2w
t4	t4	t4(min) = time_4 * T - tskew1	time_4
tA	tA	tA = (1.5 + time_ax) * T - (tco + tsui + tcable2 + tcable2 + 2*tbuf)	time_ax
t0	–	t0(min) = (time_1 + time_2 + time_9) * T	time_1, time_2r, time_9
–	–	Avoid bus contention when switching buffer on by making ton long enough	–
–	–	Avoid bus contention when switching buffer off by making toff long enough	–

1 See Figure 23-3.

23.4.3.3 Timing in Multiword DMA Mode

In multi-word DMA mode, see Figure 23-4 for read timing diagram and Figure 23-5 for write timing diagram.

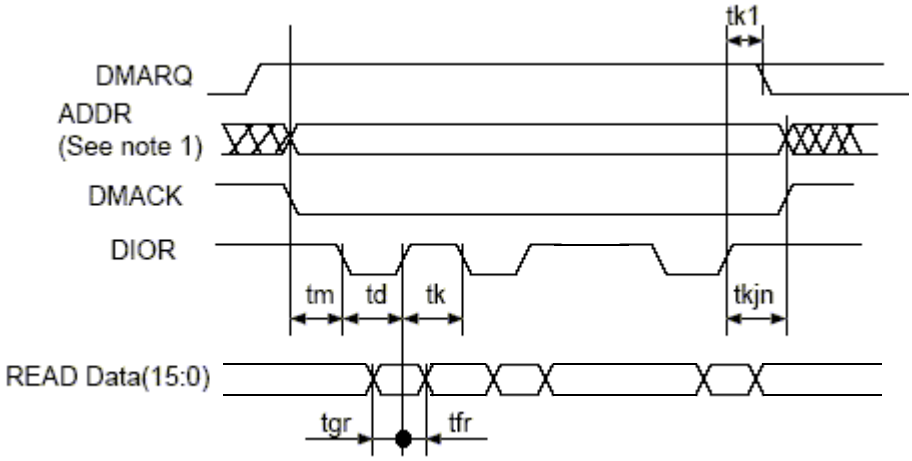


Figure 23-4. MDMA Read Timing

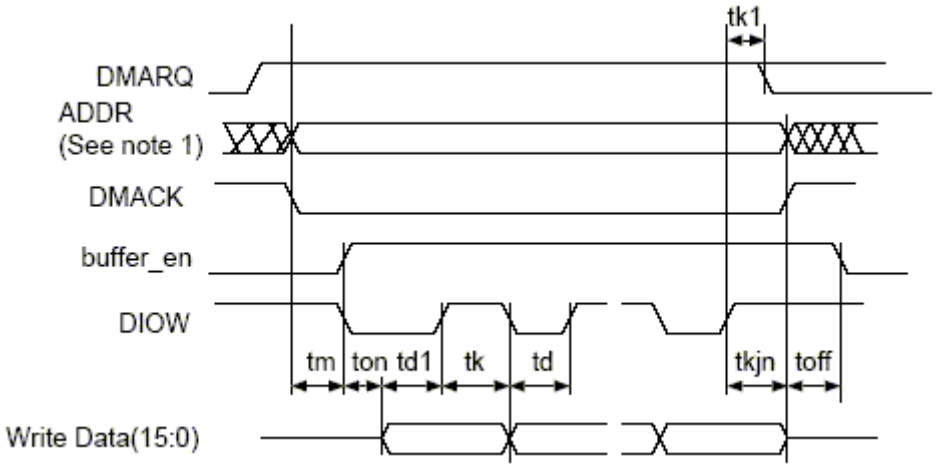


Figure 23-5. MDMA Write Timing

To meet the timing requirement, a number of timing parameters must be controlled. See Table 23-5 for details on timing parameters for MDMA read and write.

Table 23-5. Timing Parameters MDMA Read and Write

ATA Parameter	MDMA Read TIming[1] and MDMA Write TIming[2]	Value	How to Meet?
tm, ti	tm	tm(min) = ti(min) = time_m * T - (tskew1 + tskew2 + tskew5)	time_m
td	td, td1	td1(min) = td(min) = time_d * T - (tskew1 + tskew2 + tskew6)	time_d
tk	tk	tk(min) = time_k * T - (tskew1 + tskew2 + tskew6)	time_k
t0	–	t0(min) = (time_d + time_k) * T	time_d, time_k
tg(read)	tgr	tgr(min-read) = tco + tsu + tbuf + tbuf + tcable1 + tcable2 tgr(min-drive) = td - te(drive)	time_d
tf(read)	tfr	tfr(min-drive) =0k	–
tg(write)	–	tg(min-write) = time_d * T -(tskew1 + tskew2 + tskew5)	time_d

Table 23-5. Timing Parameters MDMA Read and Write (continued)

ATA Parameter	MDMA Read TIming[1] and MDMA Write TIming[2]	Value	How to Meet?
tf(write)	–	tf(min-write) = time_k * T - (tskew1 + tskew2 + tskew6)	time_k
tL	–	tL(max) = (time_d + time_k-2)*T - (tsu + tco + 2*tbuf + 2*tcable2)	time_d, time_k[3]
tn, tj	tkjn	tn= tj= tkjn = (max(time_k,. time_jn) * T - (tskew1 + tskew2 + tskew6)	time_jn
–	ton toff	ton = time_on * T - tskew1 toff = time_off * T - tskew1	–

1 See Figure 23-4.

2 See Figure 23-5.

3 tk1 in the UDMA figures equals (tk -2*T)

23.4.3.4 UDMA In Timing Diagrams

UDMA mode timing is more complicated than PIO mode or MDMA mode. In this section, timing diagrams for UDMA in are provided.

Figure 23-6 shows timing for UDMA in transfer start.

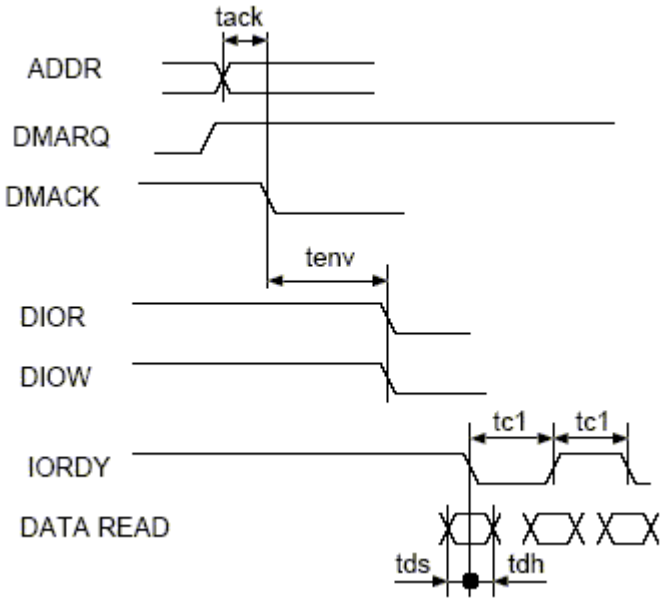


Figure 23-6. UDMA in Transfer Start Timing Diagram

Figure 23-7 shows timing for host terminating UDMA in transfer.

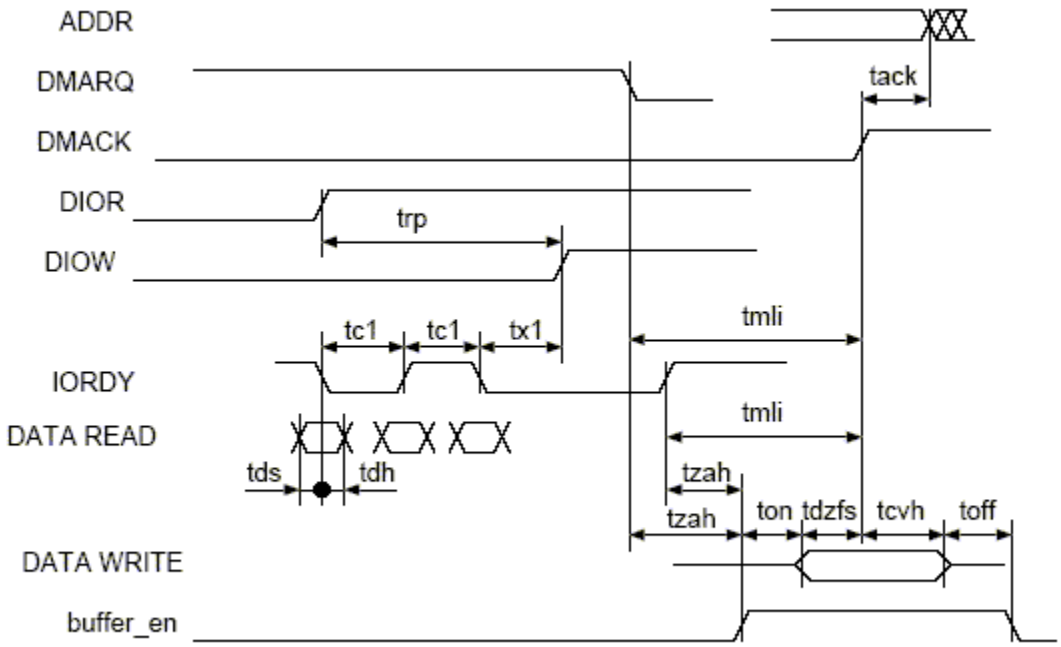


Figure 23-7. UDMA in Host Terminates Transfer

Figure 23-8 shows timing for device terminating UDMA in transfer.

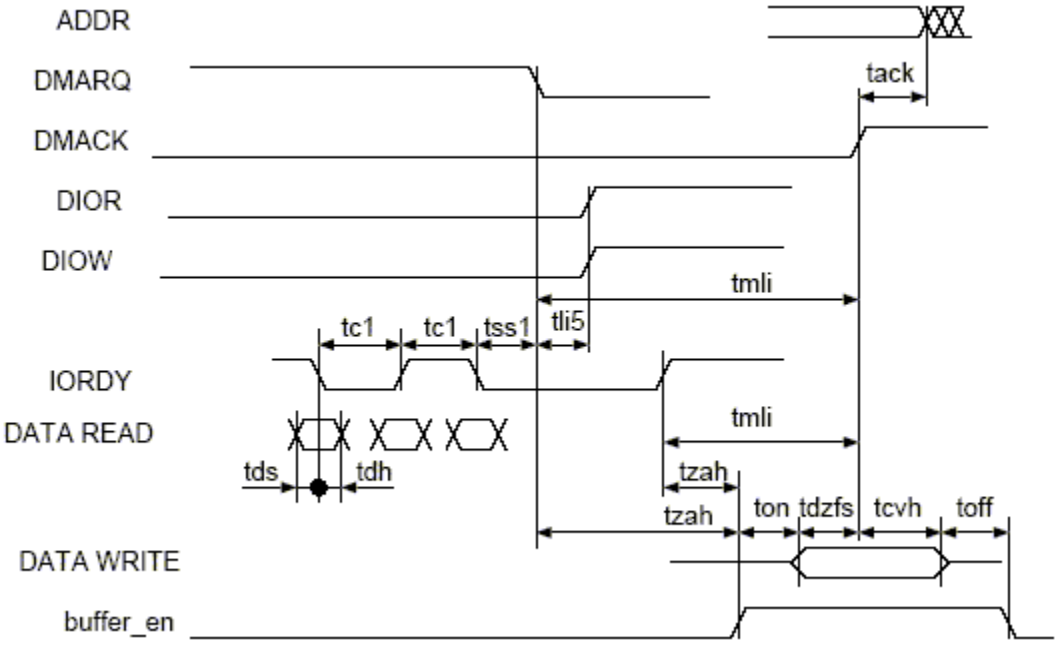


Figure 23-8. UDMA in Device Terminates Transfer

Timing parameters for UDMA in burst are listed in Table 23-6.

Table 23-6. Timing Parameters for UDMA in Burst

ATA Parameter	Spec Parameter	Value	Required Conditions
tack	tack	tack(min) = (time_ack * T) - (tskew1 + tskew2)	time_ack
tenv	tenv	tenv(min) = (time_env * T) - (tskew1 + tskew2) tenv(max) = (time_env * T) + (tskew1 + tskew2)	time_env
tds	tds1	tds - (tskew3) - ti_ds > 0	tskew3, ti_ds, ti_dh should be low enough
tdh	tdh1	tdh - (tskew3) -ti_dh > 0	
tcyc	tc1	(tcyc - tskew + TBD) > T	T big enough
trp	trp	trp(min) = time_rp * T - (tskew1 + tskew2 + tskew6)	time_rp
–	tx1[1]	(time_rp * T) - (tco + tsu + 3T + 2 *tbuf + 2*tcable2) > trfs (drive)	time_rp
tmli	tmli1	tmli1(min) = (time_mlix + 0.4) * T	time_mlix
tzah	tzah	tzah(min) = (time_zah + 0.4) * T	time_zah
tdzfs	tdzfs	tdzfs = (time_dzfs * T) - (tskew1 + tskew2)	time_dzfs
tcvh	tcvh	tcvh = (time_cvh *T) - (tskew1 + tskew2)	time_cvh
–	ton toff	ton = time_on * T - tskew1 toff = time_off * T - tskew1	–

1 There is a special timing requirement in the ATA host that requires the internal DIOW to go only high three clocks after the last active edge on the DSTROBE signal. The equation given on this line tries to capture this constraint.

2. Make ton and toff big enough to avoid bus contention.

23.4.3.5 UDMA Out Timing Diagrams

UDMA mode timing is more complicated than PIO mode or MDMA mode. In this section, timing diagrams for UDMA out are provided.

Figure 23-9 shows timing for UDMA out transfer start.

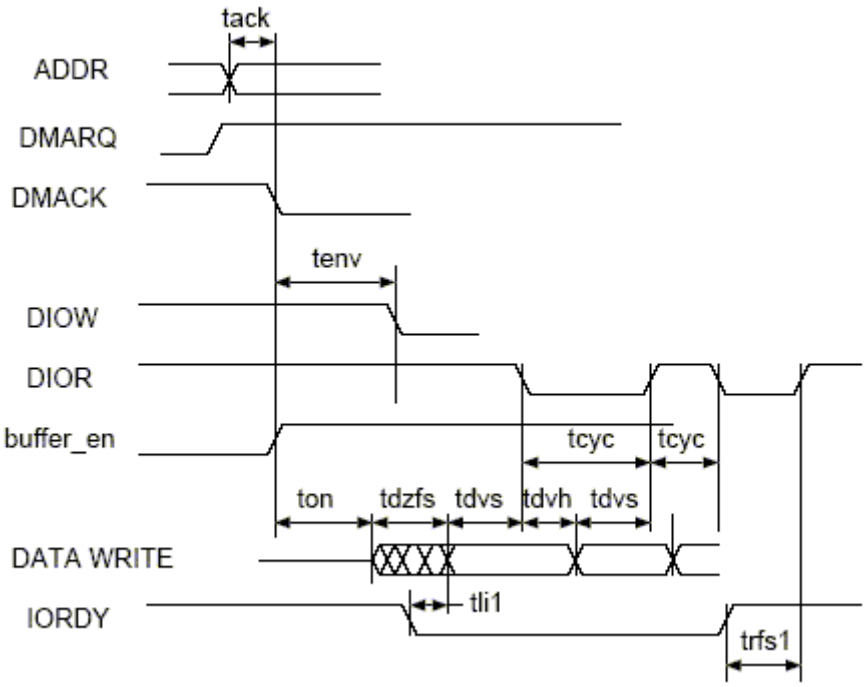


Figure 23-9. UDMA Out Transfer Start Timing Diagram

Figure 23-10 shows timing for host terminating UDMA out transfer.

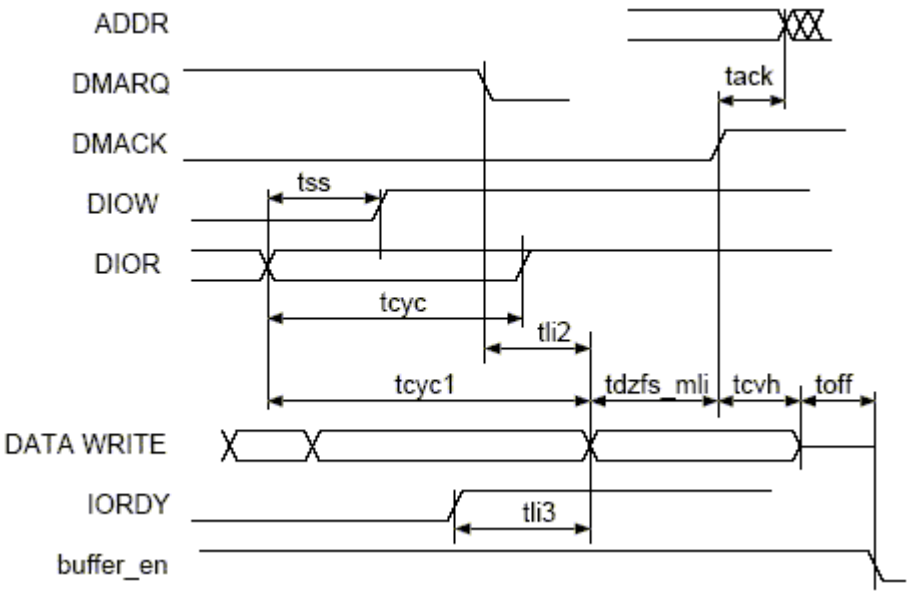


Figure 23-10. UDMA Out Host Terminates Transfer

Figure 23-11 shows timing for device terminating UDMA out transfer.

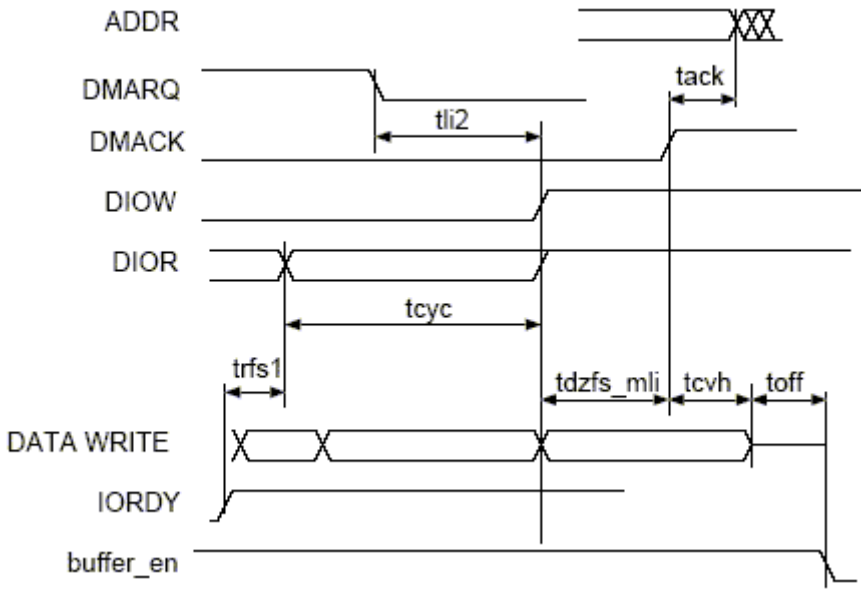


Figure 23-11. UDMA Out Device Terminates Transfer

Timing parameters for UDMA out burst are listed in Table 23-7.

Table 23-7. Timing Parameters UDMA Out Burst

ATA Parameter	Spec Parameter	Value	How to Meet?
tack	tack	tack(min) = (time_ack * T) - (tskew1 + tskew2)	time_ack
tenv	tenv	tenv(min) = (time_env * T) - (tskew1 + tskew2) tenv(max) = (time_env * T) + (tskew1 + tskew2)	time_env
tdvs	tdvs	tdvs = (time_dvs * T) - (tskew1 + tskew2)	time_dvs
tdvh	tdvh	tdvs = (time_dvh * T) - (tskew1 + tskew2)	time_dvh
tcyc	tcyc	tcyc = time_cyc * T - (tskew1 + tskew2)	time_cyc
t2cyc	–	t2cyc = time_cyc * 2 * T	time_cyc
trfs1	trfs	trfs = 1.6 * T + tsui + tco + tbuf + tbuf	–
–	tdzfs	tdzfs = time_dzfs * T - (tskew1)	time_dzfs
tss	tss	tss = time_ss * T - (tskew1 + tskew2)	time_ss
tmli	tdzfs_mli	tdzfs_mli =max(time_dzfs, time_mli) * T - (tskew1 + tskew2)	–
tli	tli1	tli1 > 0	–
tli	tli2	tli2 > 0	–
tli	tli3	tli3 > 0	–
tcvh	tcvh	tcvh = (time_cvh *T) - (tskew1 + tskew2)	time_cvh
–	ton toff	ton = time_on * T - tskew1 toff = time_off * T - tskew1	–

23.5 Memory Map and Register Definitions

Section 23.5.2, “Register Descriptions” on page 23-18 provides the detailed descriptions for all of the ATA registers.

23.5.1 Memory Map

Table 23-8 shows the ATA memory map.

Table 23-8. ATA Memory Map

Address	Register	Description	Access	Reset Value	Section/Page
MBAR2 + 0x800 (TIME_OFF)	TIME_OFF	Transceiver timing parameter. Controls toff	R/W	0x01	23.5.2.2.1/23-19
MBAR2 + 0x801 (TIME_ON)	TIME_ON	Transceiver timing parameter. Controls ton	R/W	0x01	23.5.2.2.2/23-19
MBAR2 + 0x802 (TIME_1)	TIME_1	PIO timing parameter. Controls t1	R/W	0x01	23.5.2.2.3/23-20
MBAR2 + 0x803 (TIME_2W)	TIME_2W	PIO timing parameter. Controls t2 during write cycles	R/W	0x01	23.5.2.2.4/23-20
MBAR2 + 0x804 (TIME_2R)	TIME_2R	PIO timing parameter. Controls t2 during read cycles	R/W	0x01	23.5.2.2.5/23-20
MBAR2 + 0x805 (TIME_AX)	TIME_AX	PIO timing parameter. Controls tA	R/W	0x01	23.5.2.2.6/23-21
MBAR2 + 0x80F (TIME_PIO_RDX)	TIME_PIO_RDX	PIO timing parameter. Controls trd	R/W	0x01	23.5.2.2.7/23-21
MBAR2 + 0x807 (TIME_4)	TIME_4	PIO timing parameter. Controls t4	R/W	0x01	23.5.2.2.8/23-21
MBAR2 + 0x808 (TIME_9)	TIME_9	PIO timing parameter. Controls t9	R/W	0x01	23.5.2.2.9/23-21
MBAR2 + 0x809 (TIME_M)	TIME_M	MDMA timing parameter. Controls tm	R/W	0x01	23.5.2.2.10/23-22
Address	TIME_JN	MDMA timing parameter. Controls tn and tj	R/W	0x01	23.5.2.2.11/23-22
Address	TIME_D	MDMA timing parameter. Controls td	R/W	0x01	23.5.2.2.12/23-22
Address	TIME_K	MDMA timing parameter. Controls tk	R/W	0x01	23.5.2.2.13/23-23
MBAR2 + 0x80D (TIME_ACK)	TIME_ACK	UDMA timing parameter. Controls tack	R/W	0x01	23.5.2.2.14/23-23
Address	TIME_ENV	UDMA timing parameter. Controls tenv	R/W	0x01	23.5.2.2.15/23-23
MBAR2 + 0x80F (TIME_PIO_RDX)	TIME_RPX	UDMA timing parameter. Controls trp	R/W	0x01	23.5.2.2.16/23-23
Address	TIME_ZAH	UDMA timing parameter. Controls tzah	R/W	0x01	23.5.2.2.17/23-24
MBAR2 + 0x811 (TIME_MLIX)	TIME_MLIX	UDMA timing parameter. Controls tmli	R/W	0x01	23.5.2.2.18/23-24
MBAR2 + 0x812 (TIME_DVH)	TIME_DVH	UDMA timing parameter. Controls tdvh	R/W	0x01	23.5.2.2.19/23-24
Address	TIME_DZFS	UDMA timing parameter. Controls tdzfs	R/W	0x01	23.5.2.2.20/23-25
Address	TIME_DVS	UDMA timing parameter. Controls tdvs	R/W	0x01	23.5.2.2.21/23-25
Address	TIME_CVH	UDMA timing parameter. Controls tcvh	R/W	0x01	23.5.2.2.22/23-25

Table 23-8. ATA Memory Map (continued)

Address	Register	Description	Access	Reset Value	Section/Page
Address	TIME_SS	UDMA timing parameter. Controls tss	R/W	0x01	23.5.2.2.23/23-25
Address	TIME_CYC	UDMA timing parameter. Controls tcyc and t2cyc	R/W	0x01	23.5.2.2.24/23-26
MBAR2 + 0x81C (FIFO_DATA_16)	FIFO_DATA_16	16-bit wide or 32-bit wide data port to/from FIFO	R/W	0x— — — —	23.5.2.3.1/23-26
MBAR2 + 0x818 (FIFO_DATA_32)	FIFO_DATA_32		R/W	0x— — — — — — — —	23.5.2.3.2/23-26
Address	FIFO_FILL	FIFO filling in halfwords	Read-only	0x00	23.5.2.3.3/23-27
Address	ATA_CONTROL	ATA interface control register	R/W	0x00	23.5.2.5.1/23-29
Address	INTERRUPT_PENDING	Interrupt pending register	Read-only	0x1 —	23.5.2.5.1/23-29
Address	INTERRUPT_ENABLE	Interrupt enable register	R/W	0x0 —	23.5.2.5.2/23-30
Address	INTERRUPT_CLEAR	Interrupt clear register	Write-only	0x— —	23.5.2.5.3/23-31
Address	FIFO_ALARM	FIFO alarm threshold	R/W	0x00	23.5.2.7/23-32
MBAR2 + 0x8A0 (DRIVE_DATA)	DRIVE_DATA	Drive data register	16-bit RW	–	23.5.2.7/23-32
MBAR2 + 0x8A4 (DRIVE_FEATURES)	DRIVE_FEATURES	Drive features register	R/W	–	23.5.2.7/23-32
MBAR2 + 0x8A8 (DRIVE_SECTOR_COUNT)	DRIVE_SECTOR_ COUNT	Drive sector count register	R/W	–	23.5.2.7/23-32
MBAR2 + 0x8AC (DRIVE_SECTOR_NUM)	DRIVE_SECTOR_NUM	Drive sector number register	R/W	–	23.5.2.7/23-32
MBAR2 + 0x8B0 (DRIVE_CYL_LOW)	DRIVE_CYL_LOW	Drive cylinder low register	R/W	–	23.5.2.7/23-32
MBAR2 + 0x8B4 (DRIVE_CYL_HIGH)	DRIVE_CYL_HIGH	Drive cylinder high register	R/W	–	23.5.2.7/23-32
MBAR2 + 0x8B8 (DRIVE_DEV_HEAD)	DRIVE_DEV_HEAD	Drive device head register	R/W	–	23.5.2.7/23-32
MBAR2 + 0x8BC (DRIVE_COMMAND)	DRIVE_COMMAND	Drive command register	Write-only	–	23.5.2.7/23-32
MBAR2 + 0x8C0 (DRIVE_STATUS)	DRIVE_STATUS	Drive status register	Read-only	–	23.5.2.7/23-32
MBAR2 + 0x8C4 (DRIVE_ALT_STATUS)	DRIVE_ALT_STATUS	Drive alternate status register	Read-only		23.5.2.7/23-32
MBAR2 + 0x8C8 (DRIVE_CONTROL)	DRIVE_CONTROL	Drive control register	Write-only		23.5.2.7/23-32

Table 23-9 shows the ATA register summary.

Table 23-9. ATA Register Summary

Name		7	6	5	4	3	2	1	0
MBAR2 + 0x800 (TIME_OFF)	R	TIME_OFF[7:0]							
	W								
MBAR2 + 0x801 (TIME_ON)	R	TIME_ON[7:0]							
	W								

Table 23-9. ATA Register Summary (continued)

Name		7	6	5	4	3	2	1	0
MBAR2 + 0x802 (TIME_1)	R	TIME_1[7:0]							
	W								
MBAR2 + 0x803 (TIME_2W)	R	TIME_2W[7:0]							
	W								
MBAR2 + 0x804 (TIME_2R)	R	TIME_2R[7:0]							
	W								
MBAR2 + 0x805 (TIME_AX)	R	TIME_AX[7:0]							
	W								
MBAR2 + 0x80F (TIME_PIO_RDX)	R	TIME_RDX[7:0]							
	W								
MBAR2 + 0x807 (TIME_4)	R	TIME_4[7:0]							
	W								
MBAR2 + 0x808 (TIME_9)	R	TIME_9[7:0]							
	W								
MBAR2 + 0x809 (TIME_M)	R	TIME_M[7:0]							
	W								
Address	R	TIME_JN[7:0]							
	W								
Address	R	TIME_D[7:0]							
	W								
Address	R	TIME_K[7:0]							
	W								
MBAR2 + 0x80D (TIME_ACK)	R	TIME_ACK[7:0]							
	W								
Address	R	TIME_ENV[7:0]							
	W								
Address	R	TIME_RPX[7:0]							
	W								
Address	R	TIME_ZAH[7:0]							
	W								
MBAR2 + 0x811 (TIME_MLIX)	R	TIME_MLIX[7:0]							
	W								
MBAR2 + 0x812 (TIME_DVH)	R	TIME_DVH[7:0]							
	W								
Address	R	TIME_DZFS[7:0]							
	W								
Address	R	TIME_DVS[7:0]							
	W								
Address	R	TIME_CVH[7:0]							
	W								
Address	R	TIME_SS[7:0]							
	W								

Table 23-9. ATA Register Summary (continued)

Name		7	6	5	4	3	2	1	0
Address	R	TIME_CYC[7:0]fifo_data[15:0]							
	W								

Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MBAR2 + 0x81C (FIFO_DATA_16)	R	fifo_data[15:0]															
	W																

Name		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MBAR2 + 0x818 (FIFO_DATA_32)	R	fifo_data[23:16]															
	W																
	R	fifo_data[15:0]															
	W																

Name		7	6	5	4	3	2	1	0
Address	R	FIFO_FILL[7:0]							
	W								
Address	R	fifo_rst_b	ata_rst_b	fifo_tx_en	fifo_rcv_en	dma_pending	dma_ultra_selected	dma_write	iordy_en
	W								
Address	R	ata_intrq1	fifo_underflow	fifo_overflow	controller_idle	ata_irtrq2			
	W								
Address	R	ata_intrq1	fifo_underflow	fifo_overflow	controller_idle	ata_irtrq2			
	W								
Address	R								
	W		fifo_underflow	fifo_overflow					
Address	R	FIFO_ALARM[7:0]							
	W								

23.5.2 Register Descriptions

This section contains the detailed register descriptions for the ATA registers.

23.5.2.1 Endianness

The ATA interface works both in little endian or big endian mode. The addresses of all registers are independent of endianess mode. (they remain same.) The few 16-bit and 32-bit registers represent strings of 2 or 4 bytes. The byte order in the 16-bit or 32-bit register is dependent on endianess selection.

- Little endian, 16-bit or 32-bit register:
  - bits [7:0]: byte 0
  - bits [15:8]: byte 1
  - bits [23:8]: byte 2
  - bits [31:24]: byte 3
- Big endian, 32-bit register:
  - bits [31:24]: byte 0
  - bits [23:16]: byte 1
  - bits [15:8]: byte 2
  - bits [7:0]: byte 3
- Big endian, 16-bit register:
  - bits [15:8]: byte 0
  - bits [7:0]: byte 1

23.5.2.2 Timing Registers

Registers (ata_base +$0) till (ata_base + $17) contain timing parameters. These timing parameters control the timing on the ATA bus as shown in details in the following illustrations:

- Section 23.4.3.2, “PIO Mode Timing”
- Section 23.4.3.3, “Timing in Multiword DMA Mode”
- Section 23.4.3.4, “UDMA In Timing Diagrams”
- Section 23.4.3.5, “UDMA Out Timing Diagrams”

Every timing parameter is 8-bit wide and can assume valid values between 1 and 255. Reset value is always 1.

All figures in this section show timing registers.

23.5.2.2.1 TIME_OFF Register

See Figure 23-12 for illustration of valid bits in the TIME_OFF Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x800 (TIME_OFF) Access: User read/write

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

Figure 23-12. TIME_OFF Register

23.5.2.2.2 TIME_ON Register

See Figure 23-13 for illustration of valid bits in the TIME_ON Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x801 (TIME_ON) Access: User read/write

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

Figure 23-13. TIME_ON Register

23.5.2.2.3 TIME_1 Register

See Figure for illustration of valid bits in the TIME_1 Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x802 (TIME_1) Access: User read/write

	7	6	5	4	3	2	1	0
R W	TIME_1[7:0]							
Reset	0	0	0	0	0	0	0	1

Figure 23-14. TIME_1 Register

23.5.2.2.4 TIME_2W Register

See Figure 23-15 for illustration of valid bits in the TIME_2W Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x803 (TIME_2W) Access: User read/write

	7	6	5	4	3	2	1	0
R W	TIME_2W[7:0]							
Reset	0	0	0	0	0	0	0	1

Figure 23-15. TIME_2W Register

23.5.2.2.5 TIME_2R Register

See Figure 23-16 for illustration of valid bits in the TIME_2R Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x804 (TIME_2R) Access: User read/write

	7	6	5	4	3	2	1	0
R W	TIME_2R[7:0]							
Reset	0	0	0	0	0	0	0	1

Figure 23-16. TIME_2R Register

23.5.2.2.6 TIME_AX Register

See Figure 23-17 for illustration of valid bits in the TIME_AX Register and Table 23-8 for description of the bit fields.

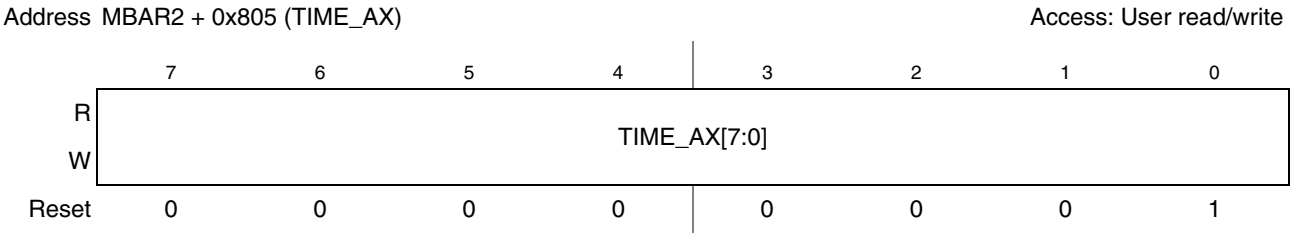


Figure 23-17. TIME_AX Register

23.5.2.2.7 TIME_PIO_RDX Register

See Figure 23-18 for illustration of valid bits in the TIME_PIO_RDX Register and Table 23-8 for description of the bit fields.

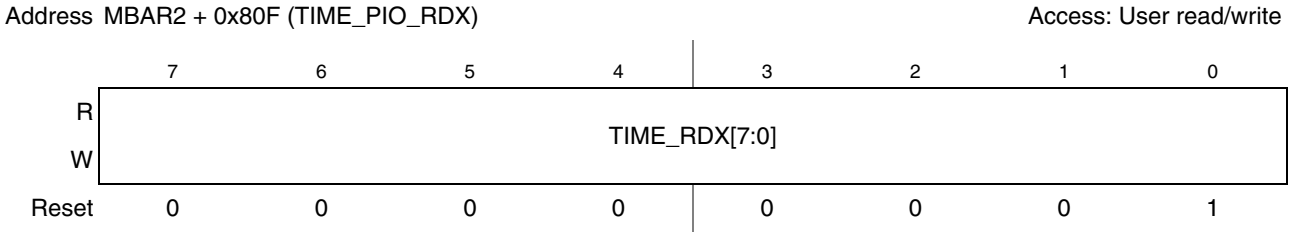


Figure 23-18. TIME_PIO_RDX Register

23.5.2.2.8 TIME_4 Register

See Figure 23-19 for illustration of valid bits in the TIME_4 Register and Table 23-8 for description of the bit fields.

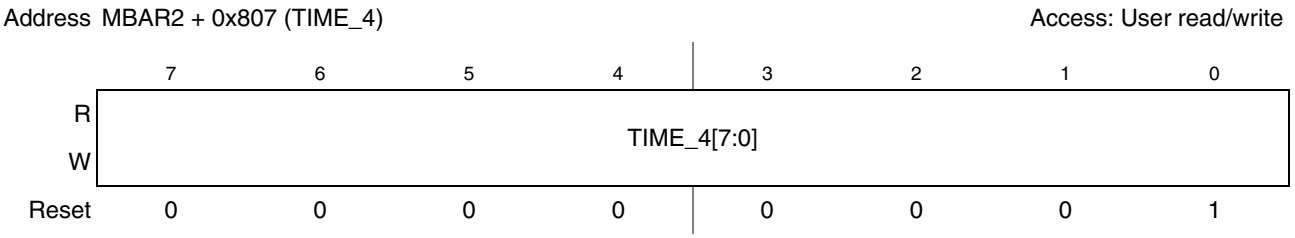


Figure 23-19. TIME_4 Register

23.5.2.2.9 TIME_9 Register

See Figure 23-20 for illustration of valid bits in the TIME_9 Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x808 (TIME_9) Access: User read/write

	7	6	5	4	3	2	1	0
R W	TIME_9[7:0]							
Reset	0	0	0	0	0	0	0	1

Figure 23-20. TIME_9 Register

23.5.2.2.10 TIME_M Register

See Figure 23-21 for illustration of valid bits in the TIME_M Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x809 (TIME_M) Access: User read/write

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

Figure 23-21. TIME_M Register

23.5.2.2.11 TIME_JN Register

See Figure 23-22 for illustration of valid bits in the TIME_JN Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x80A (TIME_JN) Access: User read/write

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

Figure 23-22. TIME_JN Register

23.5.2.2.12 TIME_D Register

See Figure 23-23 for illustration of valid bits in the TIME_D Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x80B (TIME_D) Access: User read/write

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

Figure 23-23. TIME_D Register

23.5.2.2.13 TIME_K Register

See Figure 23-24 for illustration of valid bits in the TIME_K Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x80C (TIME_K) Access: User read/write

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

Figure 23-24. TIME_K Register

23.5.2.2.14 TIME_ACK Register

See Figure 23-25 for illustration of valid bits in the TIME_ACK Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x80D (TIME_ACK) Access: User read/write

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

Figure 23-25. TIME_ACK Register

23.5.2.2.15 TIME_ENV Register

See Figure 23-26 for illustration of valid bits in the TIME_ENV Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x80E (TIME_ENV) Access: User read/write

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

Figure 23-26. TIME_ENV Register

23.5.2.2.16 TIME_RPX Register

See Figure 23-27 for illustration of valid bits in the TIME_RPX Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x80F (TIME_RPX) Access: User read/write

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

Figure 23-27. TIME_RPX Register

23.5.2.2.17 TIME_ZAH Register

See Figure 23-28 for illustration of valid bits in the TIME_ZAH Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x810 (TIME_ZAH) Access: User read/write

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

Figure 23-28. TIME_ZAH Register

23.5.2.2.18 TIME_MLIX Register

See Figure 23-29 for illustration of valid bits in the TIME_MLIX Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x811 (TIME_MLIX) Access: User read/write

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

Figure 23-29. TIME_MLIX Register

23.5.2.2.19 TIME_DVH Register

See Figure 23-30 for illustration of valid bits in the TIME_DVH Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x812 (TIME_DVH) Access: User read/write

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

Figure 23-30. TIME_DVH Register

23.5.2.2.20 TIME_DZFS Register

See Figure 23-31 for illustration of valid bits in the TIME_DZFS Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x813 (TIME_DZFS) Access: User read/write

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

Figure 23-31. TIME_DZFS Register

23.5.2.2.21 TIME_DVS Register

See Figure 23-32 for illustration of valid bits in the TIME_DVS Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x814 (TIME_DVS) Access: User read/write

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

Figure 23-32. TIME_DVS Register

23.5.2.2.22 Time_CVH Register

See Figure 23-33 for illustration of valid bits in the TIME_CVH Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x815 (TIME_CVH) Access: User read/write

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

Figure 23-33. TIME_CVH Register

23.5.2.2.23 TIME_SS Register

See Figure 23-34 for illustration of valid bits in the TIME_SS Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x816 (TIME_SS) Access: User read/write

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

Figure 23-34. TIME_SS Register

23.5.2.2.24 TIME_CYC Register

See Figure 23-35 for illustration of valid bits in the TIME_CYC Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x817 (TIME_CYC) Access: User read/write

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

Figure 23-35. TIME_CYC Register

23.5.2.3 FIFO Data Registers

23.5.2.3.1 FIFO_Data Register in 16-Bit Mode

See Figure 23-36 for illustration of valid bits in the FIFO_Data Register in 16-bit Mode and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x81C (FIFO_DATA_16) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	FIFO_DATA[15:0]															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 23-36. FIFO_Data Register In 16-bit Mode

23.5.2.3.2 FIFO_Data Register in 32-Bit Mode

See Figure 23-37 for illustration of valid bits in the FIFO_Data Register in 32-bit Mode and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x818 (FIFO_DATA_32) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	FIFO_DATA[32:16]															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	FIFO_DATA[15:0]															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 23-37. FIFO_Data Register in 32-Bit Mode

The FiFO_DATA Register is used to read or write data to the internal FIFO. It can be accessed as a 16-bit register or as a 32-bit register. Any long write to the register will put the four bytes written into the FIFO. Any word write will put the two bytes written into the FIFO. Any long read will read four bytes from the FIFO. Any word read will read two bytes from the FIFO.

23.5.2.3.3 FIFO_FILL Register

See Figure 23-38 for illustration of valid bits in the FIFO_FILL Register and Table 23-8 for description of the bit fields.

Address MBAR2 + 0x820 (FIFO_FILL) Access: User read-only

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

Figure 23-38. FIFO_FILL Register

FIFO_FILL is a read-only register. Any read to it returns the current number of halfwords present in the FIFO.

23.5.2.4 ATA_CONTROL Register

See Figure 23-39 for illustration of valid bits in the ATA Control Register and Table 23-10 for description of the bit fields.

Address MBAR2 + 0x824 (ATA_CONTROL) Access: User read/write

	7	6	5	4	3	2	1	0
R W	fifo_rst_b	ata_rst_b	fifo_tx_en	fifo_rcv_en	dma_pending	dma_ultra_selected	dma_write	iordy_en
Reset	0	0	0	0	0	0	0	0

Figure 23-39. ATA_Control Register

Table 23-10. ATA Control Register Field Descriptions

Field	Description
7 fifo_rst_b	This field controls if the internal FIFO is in reset or enabled. 0 FIFO reset 1 FIFO normal operation
6 ata_rst_b	This bit controls the level on the ata_reset_b pin, and controls the reset of the internal ata protocol engine. 0 ata_reset_b = 0, ata drive is reset, and internal protocol engine reset. 1 ata_reset_b = 1, ata drive is not reset and internal protocol engine normal operation.
5 fifo_tx_en	FIFO transmit enable. This bit controls if the FIFO will make transmit data requests to the DMA. If enabled, the FIFO will request the DMA to refill it whenever FIFO filling drops below the alarm level. 0 FIFO refill by DMA disabled 1 FIFO refill by DMA enabled
4 fifo_rcv_en	FIFO receive enable. This bit controls if the FIFO will make receive data requests to the DMA. If enabled, the FIFO will request the DMA to empty it whenever FIFO filling becomes greater or equal to the alarm level. 0 FIFO empty by DMA disabled 1 FIFO empty by DMA enabled
3 dma_pending	DMA pending bit. This bit controls if the ATA interface will respond to a DMA request originating in the drive. If this bit is asserted, the ATA interface will start a multiword DMA or ultra DMA burst whenever the drive asserts ATA_DMARQ. 0 ATA interface will not start DMA burst 1 ATA interface will start multiword DMA or ultra DMA burst whenever drive asserts dmarq
2 dma_ultra_selected	This bit indicates if a DMA burst started, the UDMA or MDMA protocol will be used. 0 Multiword DMA protocol will be used 1 Ultra DMA protocol will be used
1 dma_write	This bit indicates the data direction on any DMA burst started. 0 DMA in burst, ATA interface reads from drive 1 DMA out burst, ATA interface writes to drive
0 iordy_en	This bit indicates if the ATA_IORDY handshake will be used during PIO mode. 0 IORDY will be disregarded 1 IORDY handshake will be used

23.5.2.5 Interrupt Registers

A group of three registers control the interrupt interface from the ATA module and going to the CPU and DMA. There are two interrupts controlled by these registers:

- ipbus_int. This interrupt is controlled by bits 3, 4, 5 and 6 of the interrupt registers. It will be asserted if one of the 4 bits is set in the interrupt_pending register, while the same bit is set in the interrupt_enable register. This interrupt goes to the CPU.
- fifo_txfer_end_alarm. This interrupt is controlled by bit 7 of the interrupt registers. If ata_intrq1 is set in both the interrupt enable and interrupt pending register, fifo_txfer_end_alarm will be asserted. The goal of this interrupt is to inform the DMA that the running data transfer has ended. This interrupt goes to the smart DMA.

These three registers have mostly the same bits. If a bit is set in the interrupt pending register, its interrupt is pending, and will produce an interrupt if the same bit is set in the interrupt enable register. Some bits in the interrupt pending register are sticky bits. Writing a ‘1’ to the corresponding bit in the interrupt clear bit, will reset them.

23.5.2.5.1 Interrupt_Pending Register

See Figure 23-40 for illustration of valid bits in the Interrupt_Pending Register and Table 23-11 for description of the bit fields.

Address MBAR2 + 0x828 (INTERRUPT_PENDING) Access: User read-only

	7	6	5	4	3	2	1	0
R	ata_intrq1	fifo_underflow	fifo_overflow	controller_idle	ata_irtrq2			
W								
Reset	0[1]	0	0	1	0	–	–	–

1. Interrupts ata_intrq1 and ata_intrq2 only reset to 0 if during reset the interrupt input is low.

Figure 23-40. Interrupt_Pending Register

Table 23-11. Interrupt Pending Register Field Description

Field	Description
7 ata_intrq1	ATA interrupt request 1. This bit reflects the value of the ATA_INTRQ interrupt input. It is set in the interrupt pending register when the drive interrupt is pending, cleared otherwise. When the bit is set in the interrupt pending register, and the same bit is set in the interrupt enable register, fifo_txfer_end_alarm will be asserted, signalling the DMA the end of the transfer. The interrupt clear register has no influence on this bit.
6 fifo_underflow	FIFO underfow. This bit reports FIFO underflow. Sticky bit. It is set in the interrupt pending register when there is a FIFO underflow condition. It is cleared by writing a ‘1’ to this bit in the interrupt clear register. When the bit is set in the interrupt pending register, and the same bit is set in the interrupt enable register, ipbus_int will be active, signalling interrupt to the cpu.
5 fifo_overflow	FIFO overflow. This bit reports FIFO overflow. Sticky bit. It is set in the interrupt pending register when there is a FIFO overflow condition. It is cleared by writing a ‘1’ to this bit in the interrupt clear register. When the bit is set in the interrupt pending register, and the same bit is set in the interrupt enable register, ipbus_int will be active, signalling interrupt to the cpu.
4 controller_idle	Controller Idle. This bit reports controller idle. It is set when the ATA protocol engine is idle, there is no activity on the ATA bus. It is cleared when there is activity on the ATA bus. When the bit is set in the interrupt pending register, and the same bit is set in the interrupt enable register, ipbus_int will be active, signalling interrupt to the cpu. The interrupt clear register has no influence on this bit.

Table 23-11. Interrupt Pending Register Field Description (continued)

Field	Description
3 ata_intrq2	ATA interrupt request 2. This bit reflects the value of the ATA_INTRQ interrupt input. It is set in the interrupt pending register when the drive interrupt is pending, cleared otherwise. It has exactly same functioning as ata_intrq1, but this bit affects ipbus_int, while the other affects interrupt to the DMA. When the bit is set in the interrupt pending register, and the same bit is set in the interrupt enable register, ipbus_int will be asserted, signalling the CPU the drive is requesting attention. The interrupt clear register has no influence on this bit.
2–0 Uncommitted	N/A

23.5.2.5.2 Interrupt_Enable Register

See Figure 23-41 for illustration of valid bits in the Interrupt_Enable Register and Table 23-12 for description of the bit fields.

Address MBAR2 + 0x82C (INTERRUPT_ENABLE) Access: User read/write

	7	6	5	4	3	2	1	0
R W	ata_intrq1	fifo_underflow	fifo_overflow	controller_idle	ata_irtrq2			
Reset	0	0	0	0	0	–	–	–

Figure 23-41. Interrupt_Enable Register

Table 23-12. Interrupt Enable Register Field Description

Field	Description
7 ata_intrq1	ATA interrupt request 1. This bit reflects the value of the ATA_INTRQ interrupt input. It is set in the interrupt pending register when the drive interrupt is pending, cleared otherwise. When the bit is set in the interrupt pending register, and the same bit is set in the interrupt enable register, fifo_txfer_end_alarm will be asserted, signalling the DMA the end of the transfer. The interrupt clear register has no influence on this bit.
6 fifo_underflow	FIFO underfow. This bit reports FIFO underflow. Sticky bit. It is set in the interrupt pending register when there is a FIFO underflow condition. It is cleared by writing a ‘1’ to this bit in the interrupt clear register. When the bit is set in the interrupt pending register, and the same bit is set in the interrupt enable register, ipbus_int will be active, signalling interrupt to the cpu.
5 fifo_overflow	FIFO overflow. This bit reports FIFO overflow. Sticky bit. It is set in the interrupt pending register when there is a FIFO overflow condition. It is cleared by writing a ‘1’ to this bit in the interrupt clear register. When the bit is set in the interrupt pending register, and the same bit is set in the interrupt enable register, ipbus_int will be active, signalling interrupt to the cpu.
4 controller_idle	Controller Idle. This bit reports controller idle. It is set when the ATA protocol engine is idle, there is no activity on the ATA bus. It is cleared when there is activity on the ATA bus. When the bit is set in the interrupt pending register, and the same bit is set in the interrupt enable register, ipbus_int will be active, signalling interrupt to the cpu. The interrupt clear register has no influence on this bit.

Table 23-12. Interrupt Enable Register Field Description (continued)

Field	Description
3 ata_intrq2	ATA interrupt request 2. This bit reflects the value of the ATA_INTRQ interrupt input. It is set in the interrupt pending register when the drive interrupt is pending, cleared otherwise. It has exactly same functioning as ata_intrq1, but this bit affects ipbus_int, while the other affects interrupt to the DMA. When the bit is set in the interrupt pending register, and the same bit is set in the interrupt enable register, ipbus_int will be asserted, signalling the CPU the drive is requesting attention. The interrupt clear register has no influence on this bit.
2–0 Uncommitted	N/A

23.5.2.5.3 Interrupt_Clear Register

See Figure 23-42 for illustration of valid bits in the Interrupt_Clear Register and Table 23-13 for description of the bit fields.

Address MBAR2 + 0x830 (INTERRUPT_CLEAR) Access: User write-only

	7	6	5	4	3	2	1	0
R								
W		fifo_underflow	fifo_overflow					
Reset	–	–	–	–	–	–	–	–

Figure 23-42. Interrupt_Clear Register

Table 23-13. Interrupt Clear Register Field Description

Field	Description
7 Uncommitted	N/A
6 fifo_underflow	FIFO underfow. This bit reports FIFO underflow. Sticky bit. It is set in the interrupt pending register when there is a FIFO underflow condition. It is cleared by writing a ‘1’ to this bit in the interrupt clear register. When the bit is set in the interrupt pending register, and the same bit is set in the interrupt enable register, ipbus_int will be active, signalling interrupt to the cpu.
5 fifo_overflow	FIFO overflow. This bit reports FIFO overflow. Sticky bit. It is set in the interrupt pending register when there is a FIFO overflow condition. It is cleared by writing a ‘1’ to this bit in the interrupt clear register. When the bit is set in the interrupt pending register, and the same bit is set in the interrupt enable register, ipbus_int will be active, signalling interrupt to the cpu.
4–0 Uncommitted	N/A

23.5.2.6 FIFO Alarm Register

See Figure 23-43 for illustration of valid bits in the FIFO_Alarm Register.

Address MBAR2 + 0x834 (FIFO_ALARM) Access: User read/write

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

Figure 23-43. FIFO_Alarm Register

This register contains the threshold to generate fifo_rcv_alarm and fifo_tx_alarm to the DMA interface.

- If (fifo_tx_enable == 1 && fifo_fill < fifo_alarm): fifo_tx_alarm is set 1, request is made to DMA to refill fifo.
- If (fifo_rcv_alarm == 1 && fifo_fill >= fifo_alarm): fifo_rcv_alarm is set 1, request is made to DMA to empty fifo.

23.5.2.7 Drive Registers Connected to ATA Bus

Table 23-14. Drive Registers Connected to ATA Bus

Address	Name	Description	Access
MBAR2 + 0x8A0 (DRIVE_DATA)	drive_data	Drive data register	R/W
MBAR2 + 0x8A4 (DRIVE_FEATURES)	drive_features	Drive features register	R/W
MBAR2 + 0x8A8 (DRIVE_SECTOR_COUNT)	drive_sector_count	Drive sector count register	R/W
MBAR2 + 0x8AC (DRIVE_SECTOR_NUM)	drive_sector_num	Drive sector number register	R/W
MBAR2 + 0x8B0 (DRIVE_CYL_LOW)	drive_cyl_low	Drive cylinder low register	R/W
MBAR2 + 0x8B4 (DRIVE_CYL_HIGH)	drive_cyl_high	Drive cylinder high register	R/W
MBAR2 + 0x8B8 (DRIVE_DEV_HEAD)	drive_dev_head	Drive device head register	R/W
MBAR2 + 0x8BC (DRIVE_COMMAND)	drive_command	Drive command register	Write-only
MBAR2 + 0x8BC (DRIVE_STATUS)	drive_status	Drive status register	Read-only
MBAR2 + 0x8D8 (DRIVE_ALT_STATUS)	drive_alt_status	Drive alternate status register	Read-only
MBAR2 + 0x8D8 (DRIVE_CONTROL)	drive_control	Drive control register	Write-only

Some registers are addressable, but are not present in the ATA interface module. A list is given in Table 23-14. If a read or write access is made to one of these registers, the read or write is mapped to a PIO read or write cycle on the ATA bus, and the corresponding register in the device attached to the ATA bus is accessed.

If the drive_data register is accessed while the ATA interface operates in big endian mode, the bytes to/from the ATA bus are swapped. No swaps occur in little endian mode, nor for any other register.

23.6 Functional Description

The ATA interface provides two ways to communicate with the ATA peripherals connected to the ATA bus.

- PIO mode read/write operation to the ATA bus
- DMA transfers with the ATA bus

The operation of the peripheral is described in detail in the following sections.

23.6.1 Resetting ATA Bus

The ATA bus reset $\overline{\text{ATA\_RST}}$ is asserted whenever bit 6 ata_rst_b of register ata_control is cleared to 0. At the same time, the ATA protocol engine is reset. When this bit is set to 1, the reset is released.

23.6.2 Programming ATA Bus Timing and iordy_en

The timing the ATA interface will operate with on the ATA bus is programmable. The 24 timing registers at MBAR2 + 0x80 to MBAR2 + 0x817 are used for this. How these registers affect the timing parameters on the ATA bus, is detailed in Section 23.4, "External Signal Description." It is allowed to reprogram these registers at any time when the ATA bus is idle, so before reprogramming make sure that:

- bit dma_pending in ata_control register is cleared.
- bit controller_idle in interrupt_pending register is set.

These 2 conditions can be accomplished by first writing dma_pending to 0, then waiting until controller_idle is set, then reprogram the timing parameters. If dma_pending was 1 before the reprogramming started, it should be set again after new timing is in effect to allow the drive to finish the current DMA transfer.

It only makes sense to reprogram the bus timing in the middle of an ongoing DMA transfer when this is necessary because the operating system wants to change the bus clock period. (Dynamic voltage frequency scaling).

It is necessary to wait for controller_idle because a PIO read or write to the ATA bus terminates after the bus cycle with the CPU has been terminated. If the wait for controller_idle does not occur, the new timing values may affect a bus cycle that is still running, and cause error.

The bit iordy_en in register ata_control influences whether the ATA interface will response to the iordy signal coming from the drive. To reprogram it, same rules as for the timing registers apply: Only allowed when dma_pending is cleared, while controller_idle is set.

23.6.3 Access to ATA Bus in PIO Mode

Access to the ATA bus in PIO mode is possible after:

- ata_rst_b bit in register ata_control is set.
- Timing parameters have been programmed.

To access the drive in PIO mode, simply read or write to the correct drive register. The bus cycle will be translated to an ATA cycle, and the drive is accessed.

When drive registers are accessed while the ATA bus is in reset, the read or write is discarded, not done.

23.6.4 Using DMA Mode to Receive Data from ATA Bus

Apart from PIO mode, the ATA interface supports also MDMA and UDMA mode to transfer data. DMA mode can be used to receive data from the drive (DMA in transfer). In DMA receive mode, the protocol engine will transfer data from the drive to the FIFO using multiword DMA or ultra DMA protocol. The transfer will pause when one of following occurs:

- The FIFO is full.
- The drive deasserts its dma request signal ATA_DMARQ.
- The bit dma_pending in the ata_contol register is cleared.

When the cause of the transfer pausing is removed, the transfer restarts. The end of the transfer is signalled by the drive to the host by asserting the ATA_INTRQ signal. Alternatively, the host can read the device status register. In this register, the drive will also indicate if the transfer has ended.

The transfer of data from the FIFO into the memory is handled by the host system DMA. Whenever the FIFO filling is above the alarm threshold, the DMA should read one packet of data from the FIFO, and store this in main memory. In doing so, the DMA prevents the FIFO from getting full, and keeps the transfer from drive to FIFO running.

The steps for setting up a DMA data transfer from device to host are:

1. Make sure the ATA bus is not in reset and all timing registers are programmed.
2. Make sure the FIFO is empty by reading it until empty or by resetting it.
3. Initialize the DMA channel connected to fifo_rcv_alarm. Every time fifo_rcv_alarm is high, the DMA should read <packetsize> long ints from the FIFO, and store them to main memory. (typical packetsize is 8 longs)
4. Write 2 * <packetsize> to fifo_alarm register. In this way, FIFO will request attention to DMA when there is at least one packet ready for transfer.
5. To make the ATA ready for a DMA transfer from device to host, take the following steps:
   a) Make sure the FIFO is out of reset by setting bit fifo_rst_b to 1 in the ata control register.
   b) Program fifo_rcv_en=1 in decontrol register. This enables the FIFO to by emptied by the DMA.
   c) Program dma_pending =1, dma_write=0, ultra_mode_selected=0/1 in ata_control register. ultra_mode_selected should be 1 if you want to transfer data using UDMA mode, it should be 0 if you want to transfer data using MDMA mode.
6. Now, the host side of the DMA is ready. Send commands to the drive in PIO mode that cause it to request DMA transfer on the ATA bus. The nature of these commands is beyond the scope of this document. You should consult the ATA specification to know how to communicate with the drive.
7. When the drive now requests DMA transfer by pulling ATA_DMARQ high, the ATA interface will acknowledge with ATA_DMACK, and the transfer will start. Data is transferred automatically to the FIFO, and from there on to the host memory.
8. During the transfer, the host can monitor for end of transfer by reading some device ATA registers. These reads will cause the running DMA to pause; after the read is completed, the DMA resumes. The host can also wait unit the drive asserts ATA_INTRQ. This also indicates end of transfer.
9. On end of transfer, the host or host DMA should wait until controller_idle is set, and next read the remaining halfwords from the FIFO, and transfer these to memory.

NOTE

There may be less than <packetsize> remaining bytes, so transfer will not be automatic by the DMA.

23.6.5 Using DMA Mode to Transmit Data to ATA Bus

Apart from PIO mode, the ATA interface supports also MDMA and UDMA mode to transfer data. DMA mode can be used to transmit data to the drive (DMA out transfer). In DMA transmit mode, the protocol engine will transfer data from the FIFO to the drive using multiword DMA or ultra DMA protocol. The transfer will pause when one of following occurs:

- The FIFO is empty.
- The drive deasserts its dma request signal ATA_DMARQ.
- The bit dma_pending in the ata_contol register is cleared.

When the cause of the transfer pausing is removed, the transfer restarts. The end of the transfer is signalled by the drive to the host by asserting the ATA_INTRQ signal. Alternatively, the host can read the device status register. In this register, the drive will also indicate if the transfer has ended.

The transfer of data from the memory to the FIFO is handled by the host system DMA. Whenever the FIFO filling is below the alarm threshold, the DMA should read one packet of data from the main memory, and store this in the FIFO. In doing so, the DMA prevents the FIFO from getting empty, and keeps the transfer from FIFO to drive running.

The steps for setting up a DMA data transfer from device to host are:

1. Make sure the ATA bus is not in reset and all timing registers are programmed.
2. Make sure the FIFO is empty by reading it until empty, or by resetting it.
3. Initialize the DMA channel connected to fifo_tx_alarm. Every time fifo_tx_alarm is high, the DMA should read <packetsize> long ints from the main memory, and write them to the FIFO. (typical packetsize is 8 longs). Program the DMA such that it will not transfer more than <sectorsize> longwords in total.
4. Write FIFO_SIZE - 2 * <packetsize> to fifo_alarm register. In this way, FIFO will request attention to DMA when there is room for at least one extra packet. FIFO_SIZE should be given in halfwords. (typical 64 halfwords)
5. To make the ATA ready for a DMA transfer from host to device, perform the following steps:
   a) Make sure the FIFO is out of reset by setting bit fifo_rst_b to 1 in the ata control register.
   b) Program fifo_tx_en=1 in ata_control register. This enables the FIFO to be filled by DMA.
   c) Program dma_pending=1, dma_write=1, ultra_mode_selected=0/1 in ata_control register. ultra_mode_selected should be 1 if you want to transfer data using UDMA mode, it should be 0 if you want to transfer data using MDMA mode.
6. Now, the host side of the DMA is ready. Send commands to the drive in PIO mode that cause it to request DMA transfer on the ATA bus. The nature of these commands is beyond the scope of this document. You should consult the ATA specification to know how to communicate with the drive.

7. When the drive now requests DMA transfer by pulling ATA_DMARQ high, the ATA interface will acknowledge with ATA_DMACK, and the transfer will start. Data is transferred automatically from the FIFO, and also from host memory to FIFO.
8. During the transfer, the host can monitor for end of transfer by reading some device ATA registers. These reads will cause the running DMA to pause; after the read is completed, the DMA resumes. The host can also wait unit the drive asserts ATA_INTRQ. This also indicates end of transfer.

On end of transfer, no extra FIFO manipulations are needed.

Chapter 24
Universal Serial Bus Interface

This chapter describes the universal serial bus (USB) interface of the MCF5251. The content includes the operation, signal descriptions, host data structures, and host operations. Also provided is the device operational model and deviations from the host mode of operation.

NOTE

Portions of this chapter, "Universal Serial Bus Interface," that relate 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.

The USB interface 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.

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*
- *On-The-Go Supplement to the USB 2.0 Specification, Revision 1.0a*

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 Specification for Universal Serial Bus, Revision 1.0*
- *USB 2.0 Transceiver Macrocell Interface (UTMI) Specification, Version 1.05*

24.1 Features

The USB OTG module includes the following features:

- Complies with USB specification rev 2.0
- Supports operation as a standalone USB host controller
  - — Supports enhanced host controller interface (EHCI)
- USB device mode
- USB On-The-Go mode including host capability
- Supports high-speed (480 Mbps), full-speed (12 Mbps), and low-speed host (1.5 Mbit/s) operations
- Supports internal PHY (with UTMI+ interface)

- Supports operation as a standalone USB device
  - — Supports one upstream facing port
  - — Supports four programmable, bidirectional USB endpoints
- Host mode supports direct connect of FS/LS devices

24.2 Block Diagram

The MCF5251 implements a USB OTG module. This module may be connected to one of two external ports. Collectively the module and external ports are called the USB interface. The USB interface is shown in Figure 24-1.

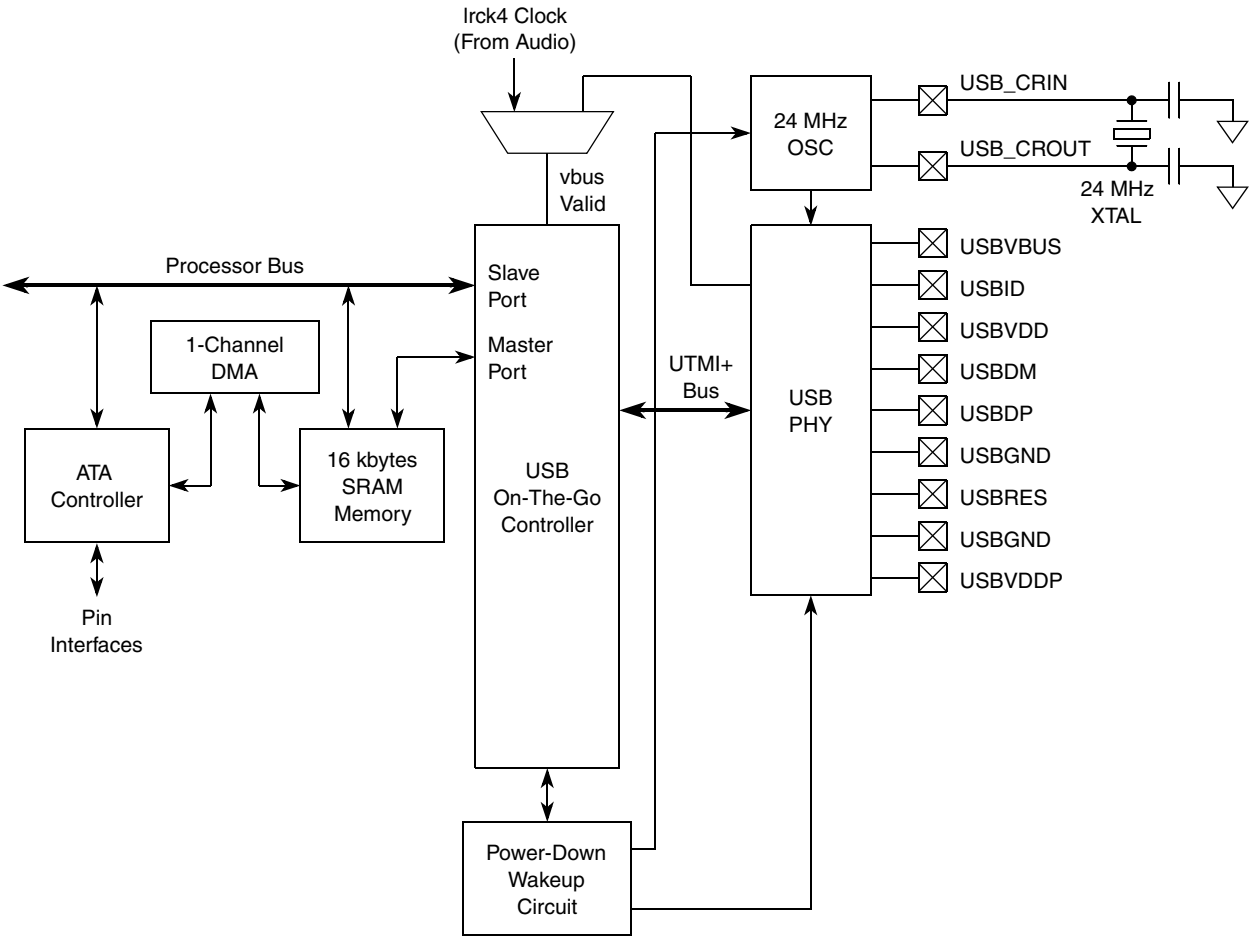


Figure 24-1. USB Interface Block Diagram

24.3 Overview

The 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 module can act as a host, as a device, or as an On-The-Go (OTG) negotiable host/device on the USB bus.

The module contains a chaining direct memory access (DMA) engine that reduces the interrupt load on the application processor, and reduces the total system bus bandwidth that must be dedicated to servicing the USB interface requirements.

To reduce bus utilization by the USB even more, the module's DMA does not have access to the CPU bus, but instead accesses a dedicated memory that can be read/written by the CPU.

24.4 Modes of Operation

The module has three basic operating modes: Host, Peripheral (device), and OTG.

The mode of operation is selectable by software. In Host mode, the module supports Low, Full and High speed USB. In Peripheral mode, only Full and High speed are supported.

24.5 External Signals

This section contains detailed descriptions of the USB interface signals.

Table 24-1 describes the external signals functionality of the USB interface.

Table 24-1. USB External Signals

Signal	I/O	Description
USBVBUS	AI	Internal UTMI+ transceiver—VBUS sensing input
USBID	I	Internal UTMI+ transceiver—ID pin
USBDN	I/O	Internal UTMI+ transceiver—DN
USBDP	I/O	Internal UTMI+ transceiver—DP
USB_CRIN	I	UTMI Internal transceiver oscillator input
USB_CROUT	O	Internal UTMI transceiver oscillator output
USBRES	AI	Internal UTMI transceiver—Bias current programming resistor

24.5.1 On-Chip Transceiver

The On-Chip transceiver is a UTMI+ specification compliant transceiver. It supports High, Full, and Low speed data transmission, in both Host and Device mode. In addition, it contains all necessary circuitry for OTG specific functionality.

Note that the USB Controller does not support Low speed operation in device mode as per USB 2.0 specification.

24.5.2 PHY Clocks

The built-in PHY has its own on-chip oscillator and PLL to generate the USB serial clocks. The oscillator needs an external 24 MHz crystal.

24.5.3 System Clock

The core logic of the USB controller is clocked with a gated copy of the system clock (CPU clock / 2). This clock can be disabled when the USB module is not in use or suspended. Note however that this clock must be enabled prior to accessing any register in the USB controller. Failing to do so will result in an unterminated bus cycle which will lock the bus.

The suspend/resume circuit that detects wake-up events on the bus remains active, even when the USB clock is turned off.

24.6 Memory Map and Register Definitions

This section provides the memory map and detailed descriptions of all USB interface registers. The memory map of the USB interface is shown in Table 24-2.

Table 24-2. USB Interface Memory Map

Offset	Register	Access	Reset	Section/Page
MBAR2 + 0x600	ID—Identification register	R	0x0040_FA05	24.6.1.1/24-5
MBAR2 + 0x604	HWGENERAL—General hardware parameters	R	0x000_0115	24.6.1.2/24-7
MBAR2 + 0x608	HWHOST—Host hardware parameters	R	0x1002_0001	24.6.1.3/24-8
MBAR2 + 0x60c	HWDEVICE—Device hardware parameters	R	0x0000_0009	24.6.1.4/24-8
MBAR2 + 0x610	HWTXBUF—TX buffer hardware parameters	R	0x8004_0604	24.6.1.5/24-9
MBAR2 + 0x614	HWRXBUF—RX buffer hardware parameters	R	0x0000_0504	24.6.1.6/24-10
MBAR2 + 0x703	CAPLENGTH—Capability Length Register	R	0x40	24.6.2.1/24-11
MBAR2 + 0x700	HCIVERSION—Host Interface Version Number	R	0x0100	24.6.2.2/24-11
MBAR2 + 0x704	HCSPARAMS—Host Control Structural parameters	R	0x0001_0011	24.6.2.3/24-12
MBAR2 + 0x708	HCCPARAMS—Host Control Capability parameters	R	0x0000_0006	24.6.2.4/24-12
MBAR2 + 0x722	DCIVERSION—Dev. Interface Version Number	R	0x0001	24.6.2.5/24-14
MBAR2 + 0x724	DCCPARAMS—Dev. Control Capability parameters	R	0x0000_0184	24.6.2.6/24-14
MBAR2 + 0x740	USBCMD—USB Command	R/W	0x0008_0000	24.6.3.1/24-15
MBAR2 + 0x744	USBSTS—USB Status	R/W	0x0000_0080	24.6.3.2/24-18
MBAR2 + 0x748	USBINTR—USB Interrupt Enable	R/W	0x0000_0000	24.6.3.3/24-20
MBAR2 + 0x74c	FRINDEX—USB Frame Index	R/W	0x0000_0000	24.6.3.4/24-21
MBAR2 + 0x754	PERIODICLISTBASE—Frame List Base Address	R/W	0x0000_0000	24.6.3.6/24-23
MBAR2 + 0x758	ASYNCLISTADDR—Next Asynchronous List Address	R/W	0x0000_0000	24.6.3.8/24-24
MBAR2 + 0x75c	TTCTRL—TT status and control	R/W	0x0000_0000	–
MBAR2 + 0x760	BURSTSIZE—Programmable DMA Burst Size	R/W	0x0000_0404	24.6.3.10/24-26
MBAR2 + 0x764	TXFILLTUNING—Host TT Xmit Pre-buffer Packet Tuning	R/W	0x0000_0000	24.6.3.11/24-27
MBAR2 + 0x780	CONFIGFLAG—Configured Flag Register	R	0x0000_0001	24.6.3.12/24-29

Table 24-2. USB Interface Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
MBAR2 + 0x784	PORTSC—Port Status and Control register	R/W	0x1c00_0004	24.6.3.13/24-29
MBAR2 + 0x7a4	OTGSC—On-The-Go Status and Control	R/W	0x0000_0020	24.6.3.14/24-34
MBAR2 + 0x7a8	USBMODE—USB Device Mode	R/W	0x0000_0000	24.6.3.15/24-37
MBAR2 + 0x7ac	ENDPOINTSETUPSTAT—Endpoint Setup Status	R/W	0x0000_0000	24.6.3.16/24-38
MBAR2 + 0x7b0	ENDPTPRIME—Endpoint Initialization	R/W	0x0000_0000	24.6.3.17/24-39
MBAR2 + 0x7b4	ENDPTFLUSH—Endpoint De-Initialize	R/W	0x0000_0000	24.6.3.18/24-40
MBAR2 + 0x7b8	ENDPTSTATUS—Endpoint Status	R	0x0000_0000	24.6.3.19/24-41
MBAR2 + 0x7bc	ENDPTCOMPLETE—Endpoint Complete	R/W	0x0000_0000	24.6.3.20/24-42
MBAR2 + 0x7c0	ENDPTCTRL0—Endpoint Control 0	R/W	0x0080_0080	24.6.3.21/24-42
MBAR2 + 0x7c4	ENDPTCTRL1—Endpoint Control 1	R/W	0x0000_0000	24.6.3.22/24-44
MBAR2 + 0x7c8	ENDPTCTRL2—Endpoint Control 2	R/W	0x0000_0000	24.6.3.22/24-44
MBAR2 + 0x7cc	ENDPTCTRL3—Endpoint Control 3	R/W	0x0000_0000	24.6.3.22/24-44

The following sections provide details about the registers in the USB memory map.

NOTE

By convention, USB registers use little-endian byte ordering. In the USB module, these are the registers from offsets 0x600 to 0x7FF.

Register addresses in this manual reflect the big-endian address.

24.6.1 Module Identification Registers

The identification registers are used to declare the slave interface presence and include a table of the hardware configuration parameters. These registers are not defined by the EHCI specification.

24.6.1.1 Identification (ID) Register

The 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.

Figure 24-2 shows the ID register.

Address MBAR2 + 0x600 Access: User read

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

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

Figure 24-2. ID Register

Table 24-3 provides bit descriptions for the ID register.

Table 24-3. ID Register Field Descriptions

Field	Description
31–24	Reserved.
23–16 REVISION	Revision number of the module.
15–14	Reserved.
13–8 NID	Ones complement version of ID[5:0].
7–6	Reserved.
5–0 ID	Configuration number. This number is set to 0x05.

24.6.1.2 General Hardware Parameters (HWGENERAL) Register

The HWGENERAL register contains parameters that define the particular implementation of the module. Figure 24-3 shows the HWGENERAL register.

Address MBAR2 + 0x604 Access: User read

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
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						SM		PHYM			PHYW		BWT	CLKC		RT
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	1

Figure 24-3. HWGENERAL Register

Table 24-4 provides bit descriptions for the HWGENERAL register.

Table 24-4. HWGENERAL Register Field Descriptions

Field	Description
31–11	Reserved.
10–9 SM	SERIAL_MODE. Always 00 indicating that the Serial Interface Engine is not present.
8–6 PHYM	PHY_MODE. Always reads 100 indicating that the USB transceiver interface mode is under software control and resets to UTMI (on-chip PHY).
5–4 PHYW	PHY Width. This field is always reads 01 indicating that the UTMI interface width is 16-bits wide.
3	Reserved. Reads as 0.
2–1	Reserved. Reads as 0b10.
0	Reserved. Always 1.

24.6.1.3 Host Hardware Parameters (HWHOST) Register

The HWHOST register provides the host hardware parameters for this implementation of the module. Figure 24-4 shows the HWHOST register.

Address MBAR2 + 0x608 Access: User read

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

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

Figure 24-4. HWHOST Register

Table 24-5 provides bit descriptions for the HWHOST register.

Table 24-5. HWHOST Register Field Descriptions

Field	Description
31–24 TTPER	Transaction translator periodic contexts. The number of supported transaction translator periodic contexts. The only legal values are 0x04 (d'4)and 0x10 (d'16).
23–16 TTASY	Transaction translator contexts. The number of transaction translator contexts.
15–4	Reserved.
3–1 NPORT	Indicates the number of ports in host mode minus 1. Always 0 for the USB OTG module.
0 HC	Always 1 indicating the module is host capable.

24.6.1.4 Device Hardware Parameters (HWDEVICE) Register—Non-EHCI

This register is not defined in the EHCI specification. The HWDEVICE register provides the device hardware parameters for this implementation.

Address MBAR2 + 0x60C Access: User read

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

Figure 24-5. HWDEVICE Register

Table 24-6 provides bit descriptions for the HWDEVICE register.

Table 24-6. HWDEVICE Register Field Descriptions

Field	Description
31–6	Reserved.
5–1 DEVEP	Device endpoints. The number of supported bidirectional endpoints; always 0x4.
0 DC	Always 1 indicating the USB OTG module is device capable.

24.6.1.5 Transmit Buffer Hardware Parameters (HWTXBUF) Register

The HWTXBUF register provides the transmit buffer parameters for this implementation of the module. Figure 24-6 shows the HWTXBUF register.

Address MBAR2 + 0x610 Access: User read

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

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

Figure 24-6. TX Buffer Hardware Parameters (HWTXBUF) Register

Table 24-7 provides bit descriptions for the HWTXBUF register.

Table 24-7. TX Buffer Hardware Parameters (HWTXBUF) Register Field Descriptions

Field	Description
31 TXLC	Reserved. Always 1 on USB OTG.
30–24	Reserved.
23–16 TXCHANADD	Transmit channel address. The number of address bits required to address one channel's worth of TX data. Always 0x4.
15–8 TXADD	Transmit address. The number of address bits for the entire TX buffer. Always 0x6.
7–0 TXBURST	Transmit burst. Indicates the number of data beats in a burst for transmit DMA data transfers. Always 0x4.

24.6.1.6 Receive Buffer Hardware Parameters (HWRXBUF) Register

The HWRXBUF register provide the receive buffer parameters for this implementation of the module. Figure 24-7 shows the HWRXBUF register.

Address MBAR2 + 0x614 Access: User read

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

Figure 24-7. RX Buffer Hardware Parameters (HWRXBUF) Register

Table 24-8 provides bit descriptions for the HWRXBUF register.

Table 24-8. RX Buffer Hardware Parameters (HWRXBUF) Register Field Descriptions

Field	Description
31–16	Reserved.
15–8 RXADD	Receive address. The number of address bits for the entire RX buffer. Always 0x5 (5).
7–0 RXBURST	Receive burst. Indicates the number of data beats in a burst for receive DMA data transfers. Always 0x4 (4).

24.6.2 Capability Registers

The capability registers specify the software limits, restrictions, and capabilities of the host/device controller implementation. Most of these registers are defined by the EHCI specification. Registers that are not defined by the EHCI specification are noted in their descriptions.

24.6.2.1 Capability Registers Length (CAPLENGTH)

This register is used as an offset to add to the register base address to find the beginning of the operational register space, that is, the location of the USBCMD register. Figure 24-8 shows the CAPLENGTH register.

Address MBAR2 + 0x703 Access: User read

	7	6	5	4	3	2	1	0
R	CAPLENGTH							
W								
Reset	0	1	0	0	0	0	0	0

Figure 24-8. Capability Registers Length (CAPLENGTH)

Table 24-9 provides bit descriptions for the CAPLENGTH register.

Table 24-9. Capability Registers Length (CAPLENGTH) Register Field Descriptions

Field	Description
7–0 CAPLENGTH	Capability registers length. Value is 0x40.

24.6.2.2 Host Controller Interface Version (HCIVERSION)

This is a two-byte register containing a BCD encoding of the EHCI revision number supported by this host controller. The most-significant byte of the register represents a major revision and the least-significant byte is the minor revision. Figure 24-9 shows the HCIVERSION register.

Address MBAR2 + 0x700 Access: User read

	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 24-9. Host Controller Interface Version (HCIVERSION) Register

Table 24-10 provides bit descriptions for the HCIVERSION register.

Table 24-10. Host Controller Interface Version (HCIVERSION) Register Field Descriptions

Field	Name	Description
15–0	–	EHCI revision number. Value is 0x0100 indicating version 1.0.

24.6.2.3 Host Controller Structural Parameters (HCSPARAMS)

This register contains structural parameters such as the number of downstream ports. Figure 24-10 shows the HCSPARAMS register.

Address MBAR2 + 0x704 Access: User read

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R					N_TT				N_PTT							PI
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	N_CC				N_PCC							PPC	N_PORTS			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1

Figure 24-10. Host Controller Structural Parameters (HCSPARAMS) Register

Table 24-11 provides bit descriptions for the HCSPARAMS register.

Table 24-11. Host Controller Structural Parameters (HCSPARAMS) Register Field Descriptions

Field	Description
31–28	Reserved.
27–24 N_TT	Number of transaction translators. This is a non-EHCI field. This field indicates the number of embedded transaction translators associated the module. This field is always 1.
23–20 N_PTT	Ports per transaction translator. This is a non-EHCI field. (EHCI defines this field as optional debug port number. DMcQ) The number of ports assigned to each transaction translator. This will always be equal to 1.
19–17	Reserved.
16 PI	Port indicators. This bit indicates whether the ports support port indicator control. Always 1. 1 The port status and control registers include a r/w field for controlling the state of the port indicator.
15–12 N_CC	Number of Companion Controllers. This field indicates the number of companion controllers associated with the controller. This field is always 0.
11–8 N_PCC	Number Ports per CC. This field indicates the number of ports supported per internal companion controller. This field is always 0.
7–5	Reserved.
4 PPC	Power Port Control. This bit indicates whether the host controller supports port power control. It is always 1. 1 Ports have power port switches.
3–0 N_PORTS	Number of Ports. This field indicates the number of physical downstream ports implemented for host applications. The value of this field determines how many port registers are addressable in the operational register. Always 0x1.

24.6.2.4 Host Controller Capability Parameters (HCCPARAMS)

This register identifies multiple mode control (time-base bit functionality) addressing capability. Figure 24-11 shows the HCCPARAMS register.

Address MBAR2 + 0x708 Access: User read

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
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	EECP								IST					ASP	PFL	ADC
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Figure 24-11. Host Control Capability Parameters (HCCPARAMS) Register

Table 24-12 provides bit descriptions for the HCCPARAMS register.

Table 24-12. Host Control Capability Parameters (HCCPARAMS) Register Field Descriptions

Field	Description
31–16	Reserved.
15–8 EECP	EHCI Extended Capabilities Pointer. This optional field indicates the existence of a capabilities list. A value of 0x00 indicates no extended capabilities are implemented. A non-zero value in this register indicates the offset in PCI configuration space of the first EHCI extended capability. The pointer value must be 0x40 or greater if implemented to maintain the consistency of the PCI header defined for this class of device. This field is always 0.
7–4 IST	Isochronous Scheduling Threshold. This field indicates, relative to the current position of the executing host controller, where the software can reliably update the isochronous schedule. When bit [7] is zero, 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. When bit [7] is a one, then the host software assumes the host controller may cache an isochronous data structure for an entire frame. This field is always 0.
3	Reserved.
2 ASP	Asynchronous Schedule Park Capability. This bit 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 field is always 1(park feature supported).
1 PFL	Programmable Frame List Flag. This bit indicates that the system software can specify and use a frame list length less that 1024 elements. 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. This field is always 1.
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

24.6.2.5 Device Controller Interface Version (DCIVERSION)

This register is 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. Figure 24-12 shows the DCIVERSION register.

Address MBAR2 0x722 Access: User read

	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 24-12. Device Interface Version (DCIVERSION) Register

Table 24-13 provides bit descriptions for the DCIVERSION register.

Table 24-13. Device Interface Version (DCIVERSION) Register Field Descriptions

Field	Description
15–0 DCIVERSION	Device interface revision number.

24.6.2.6 Device Controller Capability Parameters (DCCPARAMS) Non-EHCI

This register is not defined in the EHCI specification. This register describes the overall host/device capability of the USB OTG module. Figure 24-13 shows the DCCPARAMS register.

Address MBAR2 0x724 Access: User read

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
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								HC	DC			DEN				
W																
Reset	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0

Figure 24-13. Device Control Capability Parameters (DCCPARAMS) Register

Table 24-14 provides bit descriptions for the DCCPARAMS register.

Table 24-14. Device Control Capability Parameters (DCCPARAMS) Register Field Descriptions

Field	Description
31–9	Reserved.
8 HC	Host Capable. Always 1 indicating the controller can operate as an EHCI compatible USB 2.0 host
7 DC	Device Capable. Always 1, indicating the controller can operate as an USB 2.0 device. 0 No device capability (host only). 1 Device capability.
6–5	Reserved.
4–0 DEN	Device Endpoint Number. This field indicates the number of endpoints built into the device controller. Always 0x4.

24.6.3 Operational Registers

The operational registers are comprised of dynamic control or status registers that may be read-only, read/write, or read/write-1-to-clear. The following sections define the operational registers.

24.6.3.1 USB Command Register (USBCMD)

The module executes the command indicated in this register.

Address MBAR2 0x740 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R									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		ASPE		ASP		LR	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 24-14. USB Command Register (USBCMD) Register

Table 24-15. USB Command Register (USBCMD) Register Field Descriptions

Field	Description
31–24	Reserved.
23–16 ITC	Interrupt Threshold Control. The system software uses this field to set the maximum rate at which the module will issue interrupts. ITC contains the maximum interrupt interval measured in microframes. Valid values are shown below. 0x00 Immediate (no threshold) 0x01 1 microframe 0x02 2 microframes 0x04 4 microframes 0x08 8 microframes 0x10 16 microframes 0x20 32 microframes 0x40 40 microframes
15 FS2	See bit 3:2 below. This is a non-EHCI bit.
14 ATDTW	Add dTD TripWire. This is a non-EHCI bit,. This bit is used as a semaphore when a dTD is added to an active (primed) endpoint. This bit is set and cleared by the software. This bit shall also be cleared by the hardware when is state machine is hazard region where adding a dTD to a primed endpoint may go unrecognized. More information on the use of this bit is described in Section 24.12.2, “Device Operation,” of this manual.
13 SUTW	Setup TripWire. This is a non-EHCI bit, that is present on the USB OTG module only. This bit is used as a semaphore when the 8 bytes of setup data read extracted from a QH by the DCD. If the setup lockout mode is off (See USBMODE) then there exists a hazard when new setup data arrives and the DCD is copying setup from the QH for a previous setup packet. This bit is set and cleared by the software and will be cleared by the hardware when a hazard exists. More information on the use of this bit is described in Section 24.12.2, “Device Operation,” of this manual.
12	Reserved.
11 ASPE	Asynchronous Schedule Park Mode Enable. This bit defaults to a 0 and is R/W. The software uses this bit to enable or disable Park mode. 1 Enabled. 0 Disabled.
10	Reserved.
9–8 ASP	Asynchronous Schedule Park Mode Count. This field defaults to 00 and is R/W. It contains a count of the number of successive transactions the host controller is allowed to execute from a high-speed queue head on the Asynchronous schedule before continuing traversal of the Asynchronous schedule. Valid values are 0x1 to 0x3. The software must not write a zero to this field when Park Mode Enable is a one as this will result in undefined behavior.
7 LR	Light Host/Device Controller Reset (OPTIONAL). Not Implemented. Always 0.
6 IAA	Interrupt on Async Advance Doorbell. This bit is used as a doorbell by the software to tell the controller to issue an interrupt the next time it advances asynchronous schedule. The software must write a 1 to this bit to ring the doorbell. When the controller has evicted all appropriate cached schedule states, it sets the Interrupt on Async Advance status bit in the USBSTS register. If the Interrupt on Sync Advance Enable bit in the USBINTR register is one, then the host controller will assert an interrupt at the next interrupt threshold. The controller sets this bit to zero after it has set the Interrupt on Sync Advance status bit in the USBSTS register to one. The software should not write a one to this bit when the asynchronous schedule is inactive. Doing so will yield undefined results. This bit is used only in host mode. Writing a one to this bit when the USB OTG module is in device mode is selected will have undefined results.

Table 24-15. USB Command Register (USBCMD) Register Field Descriptions (continued)

Field	Description
5 ASE	Asynchronous Schedule Enable. This bit controls whether the controller skips processing the Asynchronous Schedule. Used only in host mode. 1 Use the ASYNCLISTADDR register to access the Asynchronous Schedule. 0 Do not process the Asynchronous Schedule.
4 PSE	Periodic Schedule Enable. This bit controls whether the controller skips processing the Periodic Schedule. Used only in host mode. 1 Use the PERIODICLISTBASE register to access the Periodic Schedule. 0 Do not process the Periodic Schedule.
3–2 FS	Frame List Size. Together with bit 15 these bits make the FS[2:0] field. This field is Read/Write only if Programmable Frame List Flag in the HCCPARAMS registers is set to 1. This field specifies the size of the frame list that controls which bits in the Frame Index Register should 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)
1 RST	Controller Reset. The software uses this bit to reset the controller. This bit is cleared by the controller when the reset process is complete. The software cannot terminate the reset process early by writing a zero to this register. Host Mode: When the software writes a one to this bit, the Host Controller resets its internal pipelines, timers, counters, state machines etc. to their initial value. Any transaction currently in progress on USB is immediately terminated. A USB reset is not driven on downstream ports. The software should not set this bit to a one when the HCHalted bit in the USBSTS register is a zero. Attempting to reset an actively running host controller will result in undefined behavior. Device Mode: When the software writes a one to this bit, the controller resets its internal pipelines, timers, counters, state machines etc. to their initial value. Any transaction currently in progress on USB is immediately terminated. Writing a one to this bit in device mode is not recommended.
0 RS	Run/Stop. Host Mode: When set to a 1, the controller proceeds with the execution of the schedule. The controller continues execution as long as this bit is set. When this bit is set to 0, the Host Controller completes the current transaction on the USB and then halts. The HC Halted bit in the status register indicates when the Host Controller has finished the transaction and has entered the stopped state. The software should not write a one to this field unless the controller is in the Halted state (that is, HCHalted in the USBSTS register is a one). Device Mode: Writing a one to this bit will cause the controller to enable a pull-up on D+ and initiate an attach event. This control bit is not directly connected to the pull-up enable, as the pull-up will become disabled upon transitioning into high-speed mode. The software should use this bit to prevent an attach event before the controller has been properly initialized. Writing a 0 to this will cause a detach event. 1 Run. 0 Stop.

24.6.3.2 USB Status Register (USBSTS)

The USB status 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. The software clears certain bits in this register by writing a 1 to them (indicated by a W1C in the bit's W cell in the figure).

Address MBAR2 0x744 Access: User read/write

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	AS	PS	RCL	HCH				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 24-15. USB Status Register (USBSTS) Register

Table 24-16. USB Status Register (USBSTS) Register Field Descriptions

Field	Description
31–16	Reserved.
15 AS	Asynchronous Schedule Status. This bit reports the current real status of the Asynchronous Schedule. The controller is not required to immediately disable or enable the Asynchronous Schedule when the software transitions the Asynchronous Schedule Enable bit in the USBCMD register. When this bit and the Asynchronous Schedule Enable bit are the same value, the Asynchronous Schedule is either enabled (1) or disabled (0). Used only in host mode. 1 Enabled. 0 Disabled.
14 PS	Periodic Schedule Status. This bit reports the current real status of the Periodic Schedule. The controller is not required to immediately disable or enable the Periodic Schedule when the software transitions the Periodic Schedule Enable bit in the USBCMD register. When this bit and the Periodic Schedule Enable bit are the same value, the Periodic Schedule is either enabled (1) or disabled (0). Used only in host mode. 1 Enabled. 0 Disabled.
13 RCL	Reclamation. This is a status bit used to detect an empty asynchronous schedule. Used only in host mode. 1 Empty asynchronous schedule. 0 Non-empty asynchronous schedule.
12 HCH	HCHalted. This bit is a zero whenever the Run/Stop bit is a one. The controller sets this bit to one after it has stopped executing because of the Run/Stop bit being set to 0, either by the software or by the Host Controller hardware (for example, internal error). Used only in host mode. 1 Halted. 0 Running.
11–9	Reserved.

Table 24-16. USB Status Register (USBSTS) Register Field Descriptions (continued)

Field	Description
8 SLI	DCSuspend. This is a non-EHCI bit that is 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. 1 Suspended. 0 Active.
7 SRI	Host mode: This is a non-EHCI status bit. In host mode, this bit will be set every 125us, provided the 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: SOF Received. When the controller detects a Start Of (micro) Frame, this bit will be set. When a SOF is extremely late, the controller will automatically set this bit to indicate that an SOF was expected. Therefore, this bit will be set roughly every 1 msec in device FS mode and every 125 msec in HS mode and will be synchronized to the actual SOF that is received. Since the controller is initialized to FS before connect, this bit will be set at an interval of 1 msec during the prelude to the connect and chirp. The software writes a 1 to this bit to clear it.
6 URI	USB Reset Received. This is a non-EHCI bit that is present on the USB OTG module only. When the controller detects a USB Reset and enters the default state, this bit will be set. The software can write a 1 to this bit to clear the USB Reset Received status bit. Used only in device mode. 1 Reset received. 0 No reset received.
5 AAI	Interrupt on Async Advance. The system software can force the controller to issue an interrupt the next time the controller advances the asynchronous schedule by writing a one to the Interrupt on Async Advance Doorbell bit in the USBCMD register. This status bit indicates the assertion of that interrupt source. Used only in host mode. 1 Async advance interrupt. 0 No async advance interrupt.
4 SEI	System Error. This bit is set whenever an error is detected on the system bus. If the System Error Enable (SEE) bit in the USBINTR is set, an interrupt will be generated. The interrupt and status bits will remain asserted until cleared by writing a 1 to this bit. Additionally, when in host mode, the RUN/STOP (RS) bit of the USBCMD register is cleared, effectively disabling the controller. For the controller in device mode, an interrupt is generated, but no other action is taken. 1 Error. 0 Normal operation.
3 FRI	Frame List Rollover. The controller sets this bit to a one when the Frame List Index rolls over from its maximum value to zero. The exact value at which the rollover occurs depends on the frame list size. For example. If the frame list size (as programmed in the Frame List Size field of the USBCMD register) is 1024, the Frame Index Register rolls over every time FRINDEX [1 3] toggles. Similarly, if the size is 512, the controller sets this bit to a one every time FHINDEX [12] toggles. Used only in host mode.
2 PCI	Host mode: Port Change Detect. The controller sets this bit to a one when on any port a Connect Status occurs, a Port Enable/Disable Change occurs, an Over Current Change occurs, or the Force Port Resume bit is set as the result of a J-K transition on the suspended port. Device mode: The controller sets this bit to a one when it enters the full or high-speed operational state. When the it exits the full or high-speed operation states due to Reset or Suspend events, the notification mechanisms are the USB Reset Received bit and the DCSuspend bits respectively. This bit is not EHCI compatible.

Table 24-16. USB Status Register (USBSTS) Register Field Descriptions (continued)

Field	Description
1 UEI (USBERRINT)	USB Error Interrupt (USBERRINT). When completion of a USB transaction results in an error condition, this bit is set by the controller. This bit is set along with the USBINT bit, if the TD on which the error interrupt occurred also had its interrupt on complete (IOC) bit set. See Section 4.15.1 in EHCI for a complete list of host error interrupt conditions. Also see Table 24-88 in this chapter for more information on device error matrix. For the controller in device mode, only resume signaling is detected, all others are ignored. 1 Error detected. 0 No error.
0 UI (USBINT)	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 Transfer Descriptor (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.

24.6.3.3 USB Interrupt Enable Register (USBINTR)

The interrupts to the software are enabled with this register. An interrupt is generated when a bit is set and the corresponding interrupt is active. The USB Status register (USBSTS) still shows interrupt sources even if they are disabled by the USBINTR register, allowing polling of interrupt events by the software.

Address MBAR2 0x748 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
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								SLE	SRE	URE	AAE	SEE	FRE	PCE	UEE	UE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-16. USB Interrupt Enable (USBINTR) Register

Table 24-17. USB Interrupt Enable (USBINTR) Register Field Descriptions

Field	Description
31–9	Reserved.
8 SLE	Sleep Enable. This is a non-EHCI bit. When this bit is a one, and the DCSuspend bit in the USBSTS register transitions, the controller will issue an interrupt. The interrupt is acknowledged by the software writing a one to the DCSuspend bit. Used only in device mode. 1 Enable. 0 Disable.

Table 24-17. USB Interrupt Enable (USBINTR) Register Field Descriptions (continued)

Field	Description
7 SRE	SOF Received Enable. This is a non-EHCI bit that is present on the USB OTG module. When this bit is a one, and the SOF Received bit in the USBSTS register is a one, the controller will issue an interrupt. The interrupt is acknowledged by the software clearing the SOF Received bit. 1 Enable. 0 Disable.
6 URE	USB Reset Enable. This is a non-EHCI bit that is present on the USB OTG module only. When this bit is a one, and the USB Reset Received bit in the USBSTS register is a one, the device controller will issue an interrupt. The interrupt is acknowledged by the software clearing the USB Reset Received bit. Used only in device mode. 1 Enable. 0 Disable.
5 AAE	Interrupt on Async Advance Enable. When this bit is a one, and the Interrupt on Async Advance bit in the USBSTS register is a one, the controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by the software clearing the Interrupt on Async Advance bit. Only used in host mode. 1 Enable. 0 Disable.
4 SEE	System Error Enable. When this bit is a one, and the System Error bit in the USBSTS register is a one, the controller will issue an interrupt. The interrupt is acknowledged by the software clearing the System Error bit. 1 Enable. 0 Disable.
3 FRE	Frame List Rollover Enable. When this bit is a one, and the Frame List Rollover bit in the USBSTS register is a one, the controller will issue an interrupt. The interrupt is acknowledged by the software clearing the Frame List Rollover bit. Used only in host mode. 1 Enable. 0 Disable.
2 PCE	Port Change Detect Enable. When this bit is a one, and the Port Change Detect bit in the USBSTS register is a one, the controller will issue an interrupt. The interrupt is acknowledged by the software clearing the Port Change Detect bit. 1 Enable. 0 Disable.
1 UEE	USB Error Interrupt Enable. When this bit is a one, and the USBERRINT bit in the USBSTS register is a one, the controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by the software clearing the USBERRINT bit in the USBSTS register. 1 Enable. 0 Disable.
0 UE	USB Interrupt Enable. When this bit is a one, and the USBINT bit in the USBSTS register is a one, the controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by the software clearing the USBINT bit. 1 Enable. 0 Disable.

24.6.3.4 Frame Index Register (FRINDEX)

In host mode, this register is used by the controller to index the periodic frame list. The register updates every 125 microseconds (once each microframe). Bits [N–3] are used to 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 the system software in the Frame List Size field in the USBCMD register.

This register must be written as a DWord. Byte writes produce-undefined results. This register cannot be written unless the controller is in the Halted state as indicated by the HCHalted bit. A write to this register while the Run/Stop hit is set produces undefined results. Writes to this register also affect the SOF value.

In device mode (), this register is read-only and, the controller updates the FRINDEX[13–3] register from the frame number indicated by the SOF marker. Whenever a SOF is received by the USB bus, FRINDEX[13–3] is checked 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 (that is, SOF for 1 msec frame). If FRINDEX[13–3] is equal to the SOF value, FRINDEX[2–0] is incremented (that is, SOF for 125 µsec microframe.)

Address MBAR2 0x74C Access: User read/write

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

Figure 24-17. USB Frame Index (FRINDEX) Register

Table 24-18. USB Frame Index (FRINDEX) Register Field Descriptions

Field	Description
31–14	Reserved.
13–0 FRINDEX	Frame index. The value in this register increments at the end of each time frame (for example, microframe). Bits [N– 3] are used for the Frame List current index. This means that each location of the frame list is accessed 8 times (frames or microframes) before moving to the next index. For the USB OTG module in device mode, the value is the current frame number of the last frame transmitted. It is not used as an index. In either mode, bits 2–0 indicate the current microframe.

Table 24-19 illustrates values of N based on the value of the Frame List Size in the USBCMD register, when used in host mode.

Table 24-19. 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

Table 24-19. FRINDEX N Values (continued)

USBCMD[FS]	Frame List Size	FRINDEX N value
100	64 elements (256 bytes)	8
101	32 elements (128 bytes)	7
110	16 elements (64 bytes)	6
111	8 elements (32 bytes)	5

24.6.3.5 Control Data Structure Segment Register (CTRLDSSEGMENT)

The CTRLDSSEGMENT register is not implemented on the MCF5251.

24.6.3.6 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 is assumed to be 4-Kbyte aligned. The contents of this register are combined with the Frame Index Register (FRINDEX) to enable the controller to step through the Periodic Frame List in sequence.

Note that this register is shared between the host and device mode functions. In host mode, it is the PERIODICLISTBASE register; in device mode, it is the DEVICEADDR register. See Section 24.6.3.7, “Device Address Register (DEVICEADDR), Non-EHCI,” for more information.

Address MBAR2 0x754 (Host Mode) Access: User read/write

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

Figure 24-18. Periodic Frame List Base Address (PERIODICLISTBASE) Register

Table 24-20. Periodic Frame List Base Address (PERIODICLISTBASE) Register Field Descriptions

Field	Description
31–12 PERBASE	Base Address. These bits correspond to memory address signal [31:12]. Used only in host mode.
11–0	Reserved.

24.6.3.7 Device Address Register (DEVICEADDR), Non-EHCI

This register is not defined in the EHCI specification. The upper seven bits of this register represent the device address. After any controller reset or a USB reset, the device address is set to the default address (0). The default address will match all incoming addresses. The software shall reprogram the address after receiving a SET_ADDRESS descriptor.

This register is shared between the host and device mode functions. In device mode, it is the DEVICEADDR register; in host mode, it is the PERIODICLISTBASE register. See Section 24.6.3.6, "Periodic Frame List Base Address Register (PERIODICLISTBASE)," for more information.

Address MBAR2 0x754 (Device Mode) Access: User read/write

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

Figure 24-19. Device Address (DEVICEADDR) Register

Table 24-21. Device Address (DEVICEADDR) Register Field Descriptions

Field	Description
31–25 USBADR	Device Address. This field corresponds to the USB device address.
24–0	Reserved.

24.6.3.8 Current Asynchronous List Address Register (ASYNCLISTADDR)

This 32-bit register contains the address of the next asynchronous queue head to be executed by the host. Bits [4–0] of this register cannot be modified by the system software and always return zeros when read.

Note that on the USB OTG module, this register is shared between the host and device mode functions. In host mode, it is the ASYNCLISTADDR register; in device mode, it is the ENDPOINTLISTADDR register. See Section 24.6.3.9, "Endpoint List Address Register (ENDPOINTLISTADDR), Non-EHCI," for more information.

Address MBAR2 0x758 (Host Mode) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	ASYBASE															
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	ASYBASE(con't)															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-20. Current Asynchronous List Address (ASYNCLISTADDR) Register

Table 24-22. Current Asynchronous List Address (ASYNCLISTADDR) Register Field Descriptions

Field	Description
31–5 ASYBASE	Link Pointer Low (LPL). These bits correspond to memory address signal [31:5]. This field may reference only a Queue Head (QH). Used only by the host controller.
4–0	Reserved.

24.6.3.9 Endpoint List Address Register (ENDPOINTLISTADDR), Non-EHCI

This register is not defined in the EHCI specification. For the module in device mode, this register contains the address of the top of the endpoint list in system memory. Bits [10–0] of this register cannot be modified by the system software and always return zeros when read. The memory structure referenced by this physical memory pointer is assumed to be 64-bytes. The queue head is actually a 48-byte structure, but must be aligned on 64-byte boundary. However, the ENDPOINTLISTADDR[EPBASE] has a granularity of 2 Kbytes, so in practice the queue head should be 2-Kbyte aligned.

This register is shared between the host and device mode functions. In device mode, it is the ENDPOINTLISTADDR register; in host mode, it is the ASYNCLISTADDR register. See Section 24.6.3.8, "Current Asynchronous List Address Register (ASYNCLISTADDR)," for more information.

Address MBAR2 0x758 (Device Mode) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	EPBASE															
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	EPBASE(con't)															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-21. Endpoint List Address (ENDPOINTLISTADDR) Register

Table 24-23. Endpoint List Address (ENDPOINTLISTADDR) Register Field Descriptions

Field	Description
31–11 EPBASE	Endpoint List Address. Address of the top of the endpoint list.
10–0	Reserved.

24.6.3.10 Master Interface Data Burst Size Register (BURSTSIZE)—Non-EHCI

This register is not defined in the EHCI specification. This register is used to control and dynamically change the burst size used during data movement on the initiator (master) interface.

Address MBAR2 0x760 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R 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	TXPBURST								RXPBURST							
Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0

Figure 24-22. Master Interface Data Burst Size (BURSTSIZE) Register

Table 24-24. Master Interface Data Burst Size (BURSTSIZE) Register Field Descriptions

Field	Description
31–16	Reserved.
15–8 TXPBURST	Programable TX Burst Length. This register 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 that 4.
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 4.

24.6.3.11 Transmit FIFO Tuning Controls Register (TXFILLTUNING)—Non-EHCI

This register is not defined in the EHCI specification. This register is used to control and dynamically change the burst size used during data movement on DMA transfers. It is used only in host mode.

The fields in this register control 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 the how much data to post into the FIFO and an estimate for how long that operation should take 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, host controller checks to ensure T_p remains before the end of the [micro]frame. If so it proceeds to pre-fill the TX FIFO. If at anytime during the pre-fill operation the time remaining the [micro]frame is $< T_s$ then the packet attempt ceases and the packet is tried 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 note 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 that will begin after the next SOF. Too many back-off events can waste bandwidth and power on the system bus and thus should be minimized (not necessarily eliminated). Back-offs can be minimized with use of the TSCHHEALTH (T_{ff}) parameter described below.

Address MBAR2 0x764 Access: User read/write

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

Figure 24-23. Transmit FIFO Tuning Controls (TXFILLTUNING) Register

Table 24-25. Transmit FIFO Tuning Controls (TXFILLTUNING) Register Field Descriptions

Field	Description
31–22	Reserved.
21–16 TXFIFOTHRES	FIFO Burst Threshold. These bits control the number of data bursts that are posted to the TX latency FIFO in host mode before the packet begins on to the bus. The minimum value is 2 and this value should be a low as possible to maximize USB performance. A higher value can be used in systems with unpredictable latency and/or insufficient bandwidth where the FIFO may underrun because the data transferred from the latency FIFO to USB occurs before it can be replenished from system memory. This value is ignored if the Stream Disable bit in USBMODE register is set. When the SDIS bit is set, the host controller behaves as if TXFIFOTHRES is set to the maximum value.
15–13	Reserved.
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 Start-Of-Frame. 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 above as T_{ff}. As an approximation, the value chosen for this register should limit the number of back-off events captured in the TXSCHHEALTH to less than 10 per second in a highly utilized bus. Choosing a value that is too high for this register is not desired as it can needlessly reduce USB utilization. The time unit represented in this register is 1.267μs when a device is connected in High-Speed Mode. The time unit represented in this register is 6.333μs when a device is connected in Low/Full-speed Mode. For most applications, TXSCHOH can be set to 4 or less. A good value to begin with is: TXFIFOTHRES * (BURSTSIZE * 4 bytes-per-word) / (40 * *TimeUnit*), always rounded to the next higher integer. *TimeUnit* is either 1.267 or 6.333 as noted earlier in this description. For example, if TXFIFOTHRES is 5 and BURSTSIZE is 8, then set TXSCHOH to 5*(8*4)/(40*1.267) = 4 for a high-speed link. If this value of TXSCHOH results in a TXSCHHEALTH count of 0 per second, try lowering the value by 1 if optimizing performance is desired. If TXSCHHEALTH exceeds 10 per second, try raising 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.

24.6.3.12 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 MBAR2 + 0x780 Access: User read

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

Figure 24-24. Configure Flag (CONFIGFLAG) Register

Table 24-26. Configure Flag (CONFIGFLAG) Register Field Descriptions

Field	Name	Description
31–0	–	Reserved. (0x0000_0001, all port routings default to this host)

24.6.3.13 Port Status and Control Registers (PORTSC)

This register is reset only when power is initially applied or in response to a controller reset. The initial conditions of a port are:

- No device connected
- Port disabled

If the port has port power control, this state remains until the software applies power to the port by setting port power to one.

For the module in device mode, the 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 the software to put the PHY into low power suspend mode and disable the PHY clock.

Address MBAR2 + 0x784 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PTS			PTW	PSPD			PFSC	PHCD	WKOC	WKDS	WLCN	PTC			
W																
Reset	0	0	0	1	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			PO	PP	LS			PR	SUSP	FPR	OCC	OCA	PEC	PE	CSC	CCS
W											Clear		Clear		Clear	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Figure 24-25. Port Status and Control (PORTSC) Register

Table 24-27. Port Status and Control (PORTSC) Register Field Descriptions

Field	Description
31–30 PTS	Port Transceiver Select. This register bit is used to control which parallel transceiver interface is selected. 00 On-Chip transceiver (UTMI+) 01 Reserved. 10 Reserved. 11 Reserved. This bit is not defined in the EHCI specification.
29	Reserved.
28 PTW	Parallel Transceiver Width. This register bit is used to control the data bus width of the parallel transceiver interface. This bit defaults to 1 after reset and is read only. 0 8-bit interface—[60 MHz] UTMI+ interface. 1 16-bit interface—[30 MHz] UTMI+ interface. PTW is valid only for UTMI mode (PTS = 00). This bit is not defined in the EHCI specification.
27–26 PSPD	Port Speed. This read-only register field indicates the speed at which the port is operating. This bit is not defined in the EHCI specification. 00 Full-speed 01 Low-speed 10 High-speed 11 Undefined
25	Reserved.
24 PFSC	Port Force Full-speed Connect. This bit is used to disable the chirp sequence that allows the port to identify itself as a HS port. 0 Allow the port to identify itself as High Speed. 1 Force the port to connect only at Full-speed. This bit is not defined in the EHCI specification.

Table 24-27. Port Status and Control (PORTSC) Register Field Descriptions (continued)

Field	Description
23 PHCD	PHY Low Power Suspend. This bit is not defined in the EHCI specification. In host mode, the PHY can be put into Low Power Suspend – when the downstream device has been put into suspend mode or when no downstream device is connected. Low power suspend is completely under the control of the software. For device mode, the PHY can be put into Low Power Suspend – when the device is not running (USBCMD Run/Stop=0b) or suspend signaling is detected on the USB. Low power suspend will be cleared automatically when the resume signaling has been detected or when forcing port resume. 0 Normal PHY operation. 1 Signal the PHY to enter low power suspend mode Reading this bit indicates the status of the PHY. **Note:** If there is no clock connected to the CLK signals, then PHCD must be set.
22 WKOC	Wake on Over-current Enable. Writing this bit to a one enables the port to be sensitive to over-current conditions as wake-up events. This field is zero if Port Power (PP) is zero. This bit is (OTG/host mode only) for use by an external power control circuit.
21 WKDS	Wake on Disconnect Enable. Writing this bit to a one enables the port to be sensitive to device disconnects as wake-up events. This field is zero if Port Power(PP) is zero or in device mode. This bit is (OTG/host mode only) for use by an external power control circuit.
20 WLCN	Wake on Connect Enable. Writing this bit to a one enables the port to be sensitive to device connects as wake-up events. This field is zero if Port Power(PP) is zero or in device mode. This bit is (OTG/host mode only) for use by an external power control circuit.
19–16 PTC	Port Test Control. Any other value than zero indicates that the port is operating in test mode. 0000 Normal operation 0001 J_STATE 0010 K_STATE 0011 SEQ_NAK 0100 Test packet 0101 FORCE_ENABLE 0110–1111 Reserved. Refer to Chapter 7 of the USB Specification Revision 2.0 [3] for details on each test mode.
15–14	Reserved
13 PO	Port Owner. This bit unconditionally goes to a 0 when the configured bit in the CONFIGFLAG register makes a 0 to 1 transition. This bit unconditionally goes to 1 whenever the Configured bit is zero. The system software uses this field to release ownership of the port to a selected the module (in the event that the attached device is not a high-speed device). The software writes a one to this bit when the attached device is not a high-speed device. A one in this bit means that an internal companion controller owns and controls the port. Port owner hand-off is not implemented in this design, therefore this bit is always 0.
12 PP	Port Power. This bit represents the current setting of the switch (0=off, 1=on). When power is not available on a port(that is, PP equals a 0), the port is non-functional and will not report attaches, detaches, etc. When an over-current condition is detected on a powered port, the PP bit in each affected port is transitioned by the host controller driver from a one to a zero (removing power from the port). This feature is implemented in the host/OTG controller (PPC = 1). For the USB OTG module in a device-only implementation port power control is not necessary, thus PPC and PP = 0.

Table 24-27. Port Status and Control (PORTSC) Register Field Descriptions (continued)

Field	Description
11–10 LS	Line Status. These bits reflect the current logical levels of the USB D+ (bit 11) and D– (bit 10) signal lines. The use of line status by the host controller driver is not necessary (unlike EHCI), because the connection of FS and LS is managed by the hardware. 00 SE0 01 J-state 10 K-state 11 Undefined
9	Reserved.
8 PR	Port Reset. In host mode, when the software writes a one to this bit the bus-reset sequence as defined in the USB Specification Revision 2.0 is started. This bit will automatically change to zero after the reset sequence is complete. This behavior is different from EHCI where the host controller driver is required to set this bit to a zero after the reset duration is timed in the driver. For the USB OTG module in device mode, this bit is a read only status bit. Device reset from the USB bus is also indicated in the USBSTS register. 1 Port is in Reset. 0 Port is not in Reset. This field is zero if Port Power(PP) is zero.
7 SUSP	Suspend In host mode: The Port Enabled bit (PE) and Suspend (SUSP) bit define the port states as follows: 0x Disable 10 Enable 11 Suspend When in suspend state, downstream propagation of data is blocked on this port, except for port reset. The blocking occurs at the end of the current transaction if a transaction was in progress when this bit was written to 1. In the suspend state, the port is sensitive to resume detection. **Note:** The bit status does not change until the port is suspended and that there may be a delay in suspending a port if there is a transaction currently in progress on the USB. The module unconditionally sets this bit to zero when the software sets the Force Port Resume bit to zero. A write of zero to this bit is ignored by the host controller. If the host software sets this bit to a one when the port is not enabled (that is, Port enabled bit is a zero) the results are undefined. This field is zero if Port Power(PP) is zero in host mode. For device mode: 1 Port in suspend state. 0 Port not in suspend state. Default. In device mode this bit is a read only status bit.

Table 24-27. Port Status and Control (PORTSC) Register Field Descriptions (continued)

Field	Description
6 FPR	Force Port Resume. This bit is not-EHCI compatible. 1 Resume detected/driven on port. 0 No resume (K-state) detected/driven on port In host mode: The software sets this bit to one to drive resume signaling. The controller sets this bit to one if a J-to-K transition is detected while the port is in the Suspend state. When this bit transitions to a one a J-to-K transition is detected, the Port Change Detect bit in the USBSTS register is also set. This bit will automatically change to zero after the resume sequence is complete. This behavior is different from EHCI where the host controller driver is required to set this bit to a zero after the resume duration is timed in the driver. **Note:** 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 a one. This bit will remain a one until the port has switched to the high-speed idle. Writing a zero has no affect because the port controller will time the resume operation clear the bit the port control state switches to HS or FS idle. This field is zero if Port Power(PP) is zero in host mode. In Device mode: After the device has been in Suspend State for 5 msec or more, the software must set this bit to one to drive resume signaling before clearing. the controller will set this bit to one if a J-to-K transition is detected while the port is in the Suspend state. The bit will be cleared when the device returns to normal operation. Also, when this bit transitions to a one because a J-to-K transition detected, the Port Change Detect bit in the USBSTS register is also set.
5 OCC	Over-current Change. The overcurrent detect function is not implemented. This bit will always read 0.
4 OCA	Over-current Active. The overcurrent detect function is not implemented. This bit will always read 0.
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). The software clears this by writing a one to it. In Device mode, the device port is always enabled. (This bit will be zero.) 1 Port disabled 0 No change This field is zero if Port Power(PP) is zero.
2 PE	Port Enabled/Disabled. In host mode ports can be enabled only by the controller as a part of the reset and enable. The software cannot enable a port by writing a one to this field. Ports can be disabled by either a fault condition (disconnect event or other fault condition) or by the host software. **Note:** 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, (0) downstream propagation of data is blocked except for reset. This field is zero if Port Power(PP) is zero in host mode. In Device Mode, the device port is always enabled. (This bit will be one).

Table 24-27. Port Status and Control (PORTSC) Register Field Descriptions (continued)

Field	Description
1 CSC	Connect Change Status. In host mode, this bit indicates a change has occurred in the port's Current Connect Status. the controller sets this bit for all changes to the port device connect status, even if the system software has not cleared an existing connect status change. For example, the insertion status changes twice before the system software has cleared the changed condition, hub hardware will be 'setting' an already-set bit (that is—the bit will remain set). The software clears this bit by writing a one to it. 1 Connect Status has changed. 0 No change This field is zero if Port Power(PP) is zero. In device mode, this bit is undefined.
0 CCS	Current Connect Status. In host mode: 1 Device is present. 0 No device present. This field is zero if Port Power(PP) is zero in host mode. In device mode: 1 Attached 0 Not attached A one indicates that the device successfully attached and is operating in either high-speed or full-speed as indicated by the High Speed Port bit in this register. A zero indicates that the device did not attach successfully or was forcibly disconnected by the software writing a zero to the Run bit in the USBCMD register. It does not state the device being disconnected or suspended.

24.6.3.14 On-The-Go Status and Control (OTGSC), Non-EHCI

This register is not defined in the EHCI specification. The USB OTG module implements one On-The-Go (OTG) Status and Control register corresponding to Port 0.

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 are de-bounced using a 1 msec time constant. Values on the status inputs that do not persist for more than 1 msec will not cause an update of the Status inputs, or cause and OTG interrupt.

Address 0x7A4 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R		DPIE	1msE	BSEIE	BSVIE	ASVIE	AVVIE	IDIE		DPIS	1msS	BSEIS	BSVIS	ASVIS	AVVIS	IDIS
W										Clear	Clear	Clear	Clear	Clear	Clear	Clear
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		DPS	1msT	BSE	BSV	ASV	AVV	ID	HABA	HADP	IDPU	DP	OT	HAAR	VC	VD
W																
Reset	0	0	0	0	0	0	0	ID	0	0	1	0	0	0	0	0

Figure 24-26. OTG Status Control (OTGSC) Register

Table 24-28. OTG Status Control (OTGSC) Register Field Descriptions

Field	Description
31	Reserved.
30 DPIE	Data Pulse Interrupt Enable. 1 Enable 0 Disable
29 1msE	1 millisecond timer Interrupt Enable. 1 Enable 0 Disable
28 BSEIE	B Session End Interrupt Enable. 1 Enable 0 Disable
27 BSVIE	B Session Valid Interrupt Enable. 1 Enable 0 Disable
26 ASVIE	A Session Valid Interrupt Enable. 1 Enable 0 Disable
25 AVVIE	A VBus Valid Interrupt Enable. 1 Enable 0 Disable
24 IDIE	USB ID Interrupt Enable. 1 Enable 0 Disable
23	Reserved.
22 DPIS	Data Pulse Interrupt Status. This bit is set when data bus pulsing occurs on DP or DN. Data bus pulsing is detected only when USBMODE.CM = Host (11) and PORTSC(0).PortPower = Off (0). The software must write a one to clear this bit.
21 1msS	1 millisecond timer Interrupt Status. This bit is set once every millisecond. The software must write a one to clear this bit.

Table 24-28. OTG Status Control (OTGSC) Register Field Descriptions (continued)

Field	Description
20 BSEIS	B Session End Interrupt Status. This bit is set when VBus has fallen below the B session end threshold. The software must write a one to clear this bit.
19 BSVIS	B Session Valid Interrupt Status. This bit is set when VBus has either risen above or fallen below the B session valid threshold (0.8 VDC). The software must write a one to clear this bit.
18 ASVIS	A Session Valid Interrupt Status. This bit is set when VBus has either risen above or fallen below the A session valid threshold (0.8 VDC). The software must write a one to clear this bit.
17 AVVIS	A VBus Valid Interrupt Status. This bit is set when VBus has either risen above or fallen below the VBus valid threshold (4.4 VDC) on an A device. The software must write a one to clear this bit.
16 IDIS	USB ID Interrupt Status. This bit is set when a change on the ID input has been detected. The software must write a one to clear this bit.
15	Reserved.
14 DPS	Data Bus Pulsing Status. 1 Pulsing detected on port 0 No pulsing on port
13 1msT	1 millisecond timer toggle. This bit toggles once per millisecond.
12 BSE	B Session End. 1 VBus is below the B session end threshold. 0 VBus is above the B session end threshold.
11 BSV	B Session Valid. 1 VBus is above the B session valid threshold. 0 VBus is below the B session valid threshold.
10 ASV	A Session Valid. 1 VBus is above the A session valid threshold. 0 VBus is below the A session valid threshold.
9 AVV	A VBus Valid. 1 VBus is above the A VBus valid threshold. 0 VBus is below the A VBus valid threshold.
8 ID	USB ID 1 B device 0 A device
7 HABA	Hardware assist B-disconnect to A-connect 0 Disabled 1 Enable automatic B-disconnect to A-connect sequence
6 HADP	Hardware assist data pulse 0 No pulse sequence started 1 Start data pulse sequence
5	Reserved.

Table 24-28. OTG Status Control (OTGSC) Register Field Descriptions (continued)

Field	Description
4 DP	Data Pulsing. 1 The pullup on DP is asserted for data pulsing during SRP. 0 The pullup on DP is not asserted.
3 OT	OTG Termination. This bit must be set when the OTG device is in device mode. 1 Enable pulldown on DN. 0 Disable pulldown on DN.
2 HAAR	Hardware assist auto-reset. 0 Disabled 1 Enable automatic reset after connect on host port
1 VC	VBUS Charge. Setting this bit causes the VBus line to be charged. This is used for VBus pulsing during SRP.
0 VD	VBUS Discharge. Setting this bit causes VBus to discharge through a resistor.

24.6.3.15 USB Mode Register (USBMODE)—Non-EHCI

This register is not defined in the EHCI specification. This register controls the operating mode of the module.

Address MBAR2 + 0x7A8 Access: User read/write

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

Figure 24-27. USB Mode (USBMODE) Register

Table 24-29. USB Mode (USBMODE) Register Field Descriptions

Field	Description
31–5	Reserved.
4 SDIS	Stream Disable. In host mode, setting this bit to a 1 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 is filled to capacity before the packet is launched onto the USB. **Note:** 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 note that in systems with high system bus utilization, setting this bit will ensure no overruns or underruns during operation, at the expense of link utilization. For those who desire optimal link performance, SDIS can be left clear, and the rules used under the description of the TXFILLTUNING register to limit underruns/overruns. 1 Active 0 Inactive In device mode (), setting this bit to a 1 disables double priming on both 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. **Note:** In High Speed Mode, all packets received will be responded to with a NYET handshake when stream disable is active.
3 SLOM	Setup Lockout Mode (). For the USB OTG module in device mode, this bit controls behavior of the setup lock mechanism. See Section 24.11.3.5, “Control Endpoint Operation Model.” 1 Setup Lockouts Off(DCD requires use of Setup Data Buffer Tripwire in USBCMD). 0 Setup Lockouts On.
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 it is necessary to switch modes, the software must reset the controller by writing to the RESET bit in the USBCMD register before reprogramming this register. 00 Idle (Default for combination host/device). 01 Reserved. 10 Device Controller (Default for device only controller). 11 Host Controller (Default for host only controller). The USB OTG module defaults to the idle state and needs to be initialized to the desired operating mode after reset.

24.6.3.16 Endpoint Setup Status Register (ENDPTSETUPSTAT)—Non-EHCI

This register is not defined in the EHCI specification. This register contains the endpoint setup status. It is used only in device mode.

Address MBAR2 0x7AC Access: User read/write

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

Figure 24-28. Endpoint Setup Status (ENDPTSETUPSTAT) Register

Table 24-30. Endpoint Setup Status (ENDPTSETUPSTAT) Register Field Descriptions

Field	Description
31–4	Reserved.
3–0 ENDPTSETUPSTAT	Setup Endpoint Status. For every setup transaction that is received, a corresponding bit in this register is set. The software must clear or acknowledge the setup transfer by writing a one to a respective bit after it has read the setup data from 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 is engaged. This register is used only in device mode.

24.6.3.17 Endpoint Initialization Register (ENDPTPRIME)—Non-EHCI

This register is not defined in the EHCI specification. This register is used to initialize endpoints. It is used by the USB OTG module only in device mode.

Address MBAR2 + 0x7B0 Access: User read/write

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

Figure 24-29. Endpoint Initialization (ENDPTPRIME) Register

Table 24-31. Endpoint Initialization (ENDPTPRIME) Register Field Descriptions

Field	Description
31–20	Reserved.
19–16 PETB	Prime endpoint transmit buffer. For each endpoint, a corresponding bit is used to request that a buffer prepared for a transmit operation in order to respond to a USB IN/INTERRUPT transaction. The software should write a one to the corresponding bit when posting a new transfer descriptor to an endpoint. The hardware will automatically use this bit to begin parsing for a new transfer descriptor from the queue head and prepare a transmit buffer. The hardware will clear this bit when the associated endpoint(s) is (are) successfully primed. PETB[3] (bit 19 of the register) corresponds to endpoint 3. **Note:** These bits will be momentarily set by the hardware during hardware re-priming operations when a dTD is retired, and the dQH is updated.
15–4	Reserved.
3–0 PERB	Prime endpoint receive buffer. For each endpoint, a corresponding bit is used to request a buffer prepare for a receive operation in order to respond to a USB OUT transaction. The software should write a one to the corresponding bit whenever posting a new transfer descriptor to an endpoint. The hardware will automatically use this bit to begin parsing for a new transfer descriptor from the queue head and prepare a receive buffer. The hardware will clear this bit when the associated endpoint(s) is (are) successfully primed. **Note:** These bits will be momentarily set by the hardware during hardware re-priming operations when a dTD is retired, and the dQH is updated.

24.6.3.18 Endpoint Flush Register (ENDPTFLUSH), Non-EHCI

This register is not defined in the EHCI specification. This register is used by the USB OTG module only in device mode.

Address MBAR2 + 0x7B4 Access: User read/write

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

Figure 24-30. Endpoint Flush (ENDPTFLUSH) Register

Table 24-32. Endpoint Flush (ENDPTFLUSH) Register Field Descriptions

Field	Description
31–20	Reserved.
19–16 FETB	Flush endpoint transmit buffer. Writing a one to a bit(s) in this register will cause the associated endpoint(s) to clear any primed buffers. If a packet is in progress for one of the associated endpoints, then that transfer will continue until completion. The hardware will clear this register after the endpoint flush operation is successful.

Table 24-32. Endpoint Flush (ENDPTFLUSH) Register Field Descriptions (continued)

Field	Description
15–4	Reserved.
3–0 FERB	Flush endpoint receive buffer. Writing a one to a bit(s) will cause the associated endpoint(s) to clear any primed buffers. If a packet is in progress for one of the associated endpoints, then that transfer will continue until completion. The hardware will clear this register after the endpoint flush operation is successful.

24.6.3.19 Endpoint Status Register (ENDPTSTATUS), Non-EHCI

This register is not defined in the EHCI specification. This register is used by the USB OTG module only in device mode.

Address MBAR2 + 0x7B8 Access: User read

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

Figure 24-31. Endpoint Status (ENDPTSTATUS) Register

Table 24-33. Endpoint Status (ENDPTSTATUS) Register Field Descriptions

Field	Description
31–20	Reserved.
19–16 ETBR	Endpoint transmit buffer ready. One bit for each endpoint indicates status of the respective endpoint buffer. This bit is set by the hardware as a response to receiving a command from a corresponding bit in the ENDPTPRIME register. There will always be a delay between setting a bit in the ENDPTPRIME register and endpoint indicating ready. This delay time varies based upon the current USB traffic and the number of bits set in the ENDPTPRIME register. Buffer ready is cleared by USB reset, by the USB DMA system, or through the ENDPTFLUSH register. ETBR[3] (bit 19 of the register) corresponds to endpoint 3. **Note:** These bits will be momentarily cleared by the hardware during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated.
15–4	Reserved.
3–0 ERBR	Endpoint receive buffer ready. One bit for each endpoint indicates status of the respective endpoint buffer. This bit is set by the hardware as a response to receiving a command from a corresponding bit in the ENDPTPRIME register. There will always be a delay between setting a bit in the ENDPTPRIME register and endpoint indicating ready. This delay time varies based upon the current USB traffic and the number of bits set in the ENDPTPRIME register. Buffer ready is cleared by USB reset, by the USB DMA system, or through the ENDPTFLUSH register. ERBR[3] (bit 3 of the register) corresponds to endpoint 3. **Note:** These bits will be momentarily cleared by the hardware during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated.

24.6.3.20 Endpoint Complete Register (ENDPTCOMPLETE), Non-EHCI

This register is not defined in the EHCI specification. This register is used by the USB OTG module only in device mode.

Address MBAR2 + 0x7BC Access: User read/write

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

Figure 24-32. Endpoint Complete (ENDPTCOMPLETE) Register

Table 24-34. Endpoint Complete (ENDPTCOMPLETE) Register Field Descriptions

Field	Description
31–20	Reserved.
19–16 ETCE	Endpoint transmit complete event. Each bit indicates a transmit event (IN/INTERRUPT) occurred and the software should read the corresponding endpoint queue to determine the endpoint status. If the corresponding IOC bit is set in the Transfer Descriptor, then this bit will be set simultaneously with the USBINT. Writing a one will clear the corresponding bit in this register. ETCE[3] (bit 19 of the register) corresponds to endpoint 3.
15–4	Reserved.
3–0 ERCE	Endpoint receive complete event. Each bit indicates a received event (OUT/SETUP) occurred and the software should read the corresponding endpoint queue to determine the transfer status. If the corresponding IOC bit is set in the Transfer Descriptor, then this bit will be set simultaneously with the USBINT. Writing a one will clear the corresponding bit in this register. ERCE[3] (bit 3 of the register) corresponds to endpoint 3.

24.6.3.21 Endpoint Control Register 0 (ENDPTCTRL0), Non-EHCI

This register is not defined in the EHCI specification. Every device will implement endpoint 0 as a control endpoint.

Address MBAR2 + 0x7C0 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R									TXE				TXT			TXS
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R									RXE				RXT			RXS
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Figure 24-33. Endpoint Control 0 (ENDPTCTRL0) Register

Table 24-35. Endpoint Control 0 (ENDPTCTRL0) Register Field Descriptions

Field	Description
31–24	Reserved.
23 TXE	TX Endpoint Enable. Endpoint zero is always enabled. 1 Enable
22–20	Reserved.
19–18 TXT	TX Endpoint type. Endpoint zero is always a control endpoint (00).
17	Reserved.
16 TXS	TX Endpoint Stall. The software can write a one to this bit to force the endpoint to return a STALL handshake to the Host. It will continue returning STALL until the bit is cleared by the software or it will automatically be cleared upon receipt of a new SETUP request. 1 Endpoint Stalled 0 Endpoint OK
15–8	Reserved.
7 RXE	RX Endpoint Enable. Endpoint zero is always enabled. 1 Enabled.
6–4	Reserved.
3–2 RXT	RX Endpoint type. Endpoint zero is always a control endpoint (00).
1	Reserved.
0 RXS	RX Endpoint Stall. The software can write a one to this bit to force the endpoint to return a STALL handshake to the Host. It will continue returning STALL until the bit is cleared by the software or it will automatically be cleared upon receipt of a new SETUP request. 1 Endpoint Stalled 0 Endpoint OK

24.6.3.22 Endpoint Control Register *n* (ENDPTCTRL*n*), Non-EHCI

These registers are not defined in the EHCI specification. There is an ENDPTCTRL*n* register of each endpoint in a device.

Address MBAR2 + 0x7C4 (ENDPTCTRL1)
MBAR2 + 0x7C8 (ENDPTCTRL2)
MBAR2 + 0x7CC (ENDPTCTRL3)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R									TXE	TXR	TXI		TXT		TXD	TXS
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									RXE	RXR	RXI		RXT		RXD	RXS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-34. Endpoint Control 1 to 3(ENDPTCTRL*n*) Register

Table 24-36. Endpoint Control 1 to 3 (ENDPTCTRL*n*) Register Field Descriptions

Field	Description
31–24	Reserved.
23 TXE	TX endpoint enable. 1 Enabled 0 Disabled
22 TXR	TX data toggle reset. Whenever a configuration event is received for this Endpoint, the software must write a one to this bit in order to synchronize the data PID's between the Host and device.
21 TXI	TX data toggle inhibit. This bit is used for test only and should always be written as zero. Writing a one to this bit will cause this endpoint to ignore the data toggle sequence and always transmit DATA0 for a data packet. 1 PID Sequencing Disabled 0 PID Sequencing Enabled
20	Reserved.
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.

Table 24-36. Endpoint Control 1 to 3 (ENDPTCTRL*n*) Register Field Descriptions (continued)

Field	Description
16 TXS	TX endpoint stall. This bit will be set automatically upon receipt of a SETUP request if this Endpoint is not configured as a Control Endpoint. It will be cleared automatically upon receipt of a SETUP request if this Endpoint is configured as a Control Endpoint. The software can write a one to this bit to force the endpoint to return a STALL handshake to the Host. It will continue to returning STALL until this bit is either cleared by the software or automatically cleared as above. 1 Endpoint Stalled 0 Endpoint OK
15–8	Reserved.
7 RXE	RX endpoint enable. 1 Enabled 0 Disabled
6 RXR	RX data toggle reset. Whenever a configuration event is received for this Endpoint, the software must write a one to this bit in order to synchronize the data PID's between the Host and device.
5 RXI	RX data toggle inhibit. This bit is used for test only and should always be written as zero. Writing a one to this bit will cause this endpoint to ignore the data toggle sequence and always accept data packets regardless of their data PID. 1 PID Sequencing Enabled 0 PID Sequencing Disabled
4	Reserved.
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 will be set automatically upon receipt of a SETUP request if this Endpoint is not configured as a Control Endpoint. It will be cleared automatically upon receipt a SETUP request if this Endpoint is configured as a Control Endpoint, The software can write a one to this bit to force the endpoint to return a STALL handshake to the Host. It will continue to returning STALL until this bit is either cleared by the software or automatically cleared as above, 1 Endpoint Stalled 0 Endpoint OK

24.7 Functional Description

The USB module can be broken down into functional sub-blocks described below.

24.7.1 DMA Engine

The DMA Engine interfaces internally to dedicated DMA cache memory and has no access to main memory. It is responsible for moving all of the data to be transferred over the USB between the module and buffers in DMA cache memory. Like the system interface block the DMA engine block uses a simple synchronous bus signaling protocol.

The DMA controllers must access both control information and packet data from DMA cache memory. The control information is contained in link list based queue structures. The DMA controllers have state machines that are able to parse data structures defined in the EHCI specification. In host mode, the data structures are EHCI compliant and represent queues of transfers to be 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, the data structures designed to be similar to those in the EHCI specification and are used to allow device responses to be queued for each of the active pipes in the device.

The DMA controller can access only the DMA_CACHE memory. Therefore, all data and data structures that are read/written by the DMA engine must be reside in this memory. The USB module has priority on this memory and will get access 1 clock cycle after the request.

24.7.2 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 is maintained in each direction through the buffer memory. In device mode, multiple FIFO channels are maintained for each of the active endpoints in the system.

In host mode, the module uses a 256-byte TX buffer and a 128-byte RX buffer. Device operation uses a single 128-byte RX buffer and a 64-byte TX buffer for each endpoint.

24.7.3 PHY Interface

The module interfaces to the internal PHY. The primary function of the port controller block is to isolate the rest of the module from the transceiver, and to move all of the transceiver signaling into the primary clock domain of the module. This allows the module to run synchronously with the system processor and it's associated resources.

24.8 Host Data Structures

This section defines the interface data structures used to communicate control, status, and data between HCD (software) and the Enhanced Host Controller (hardware). The data structure definitions in this section support a 32-bit memory buffer address space. The interface consists of a Periodic Schedule, Periodic Frame List, Asynchronous Schedule, Isochronous Transaction Descriptors, Split-transaction Isochronous Transfer Descriptors, Queue Heads, and Queue Element Transfer Descriptors.

The periodic frame list is the root of all periodic (isochronous and interrupt transfer type) support for the host controller interface. The asynchronous list is the root for all the bulk and control transfer type support. Isochronous data streams are managed using Isochronous Transaction Descriptors. Isochronous split-transaction data streams are managed with Split-transaction Isochronous Transfer Descriptors. All Interrupt, Control, and Bulk data streams are managed via queue heads and Queue Element Transfer Descriptors. These data structures are optimized to reduce the total memory footprint of the schedule and to reduce (on average) the number of memory accesses needed to execute a USB transaction.

Note that the software must ensure that no interface data structure reachable by the EHCI host controller spans a 4K-page boundary.

The data structures defined in this section are (from the host controller's perspective) a mix of read-only and read/writable fields. The host controller must preserve the read-only fields on all data structure writes.

24.8.1 Periodic Frame List

Figure 24-35 shows the organization of the periodic schedule. This schedule is for all periodic transfers (isochronous and interrupt). The periodic schedule is referenced from the operational registers space using the PERIODICLISTBASE address register and the FRINDEX register. The periodic schedule is based on an array of pointers called the Periodic Frame List. The PERIODICLISTBASE address register is combined with the FRINDEX register to produce a memory pointer into the frame list. The Periodic Frame List implements a sliding window of work over time.

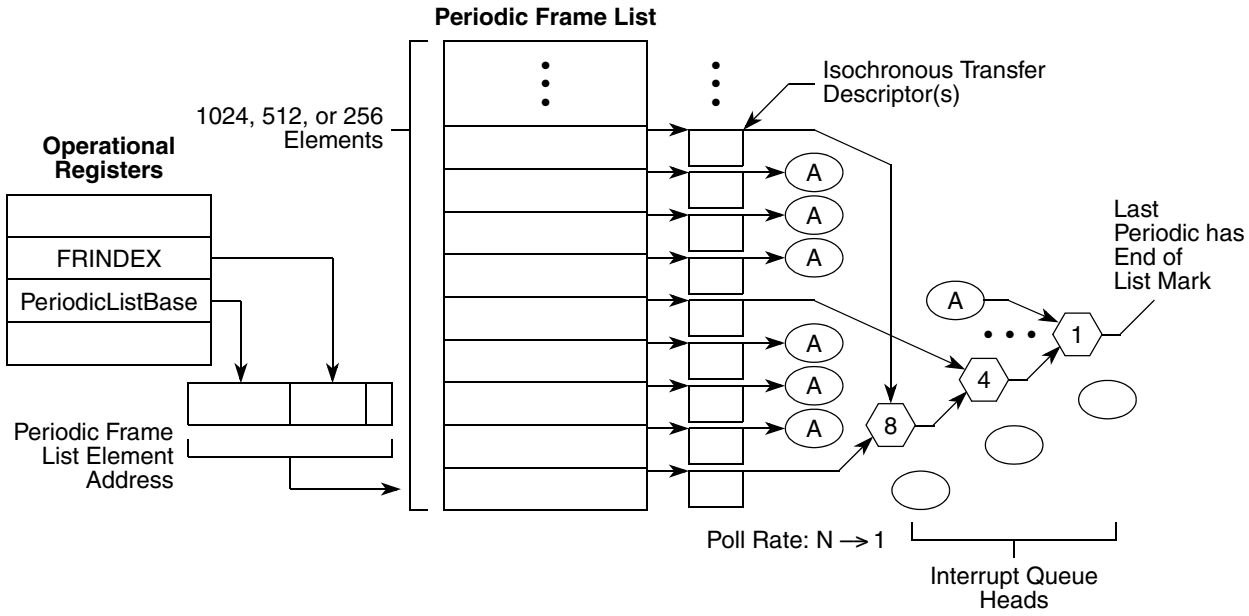


Figure 24-35. Periodic Schedule Organization

Split transaction Interrupt, Bulk and Control are also managed using queue heads and queue element transfer descriptors.

The periodic frame list is a 4K-page aligned array of Frame List Link pointers. The length of the frame list may be programmable. The programmability of the periodic frame list is exported to the system software via the HCCPARAMS register. If non-programmable, the length is 1024 elements. If programmable, the length can be selected by the system software as one of 256, 512, or 1024 elements. An implementation must support all three sizes. Programming the size (that is, the number of elements) is accomplished by the system software writing the appropriate value into Frame List Size field in the USBCMD register.

Frame List Link pointers direct the host controller to the first work item in the frame's periodic schedule for the current micro-frame. The link pointers are aligned on DWord boundaries within the Frame List. Figure 24-36 shows the format for the Frame List Link Pointer.

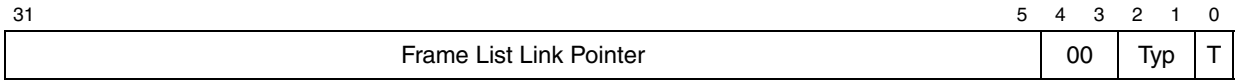


Figure 24-36. Frame List Link Pointer Format

Frame List Link pointers always reference memory objects that are 32-byte aligned. The referenced object may be an isochronous transfer descriptor for high-speed devices, a split-transaction isochronous transfer descriptor (for full-speed isochronous endpoints), or a queue head (used to support high-, full- and low-speed interrupt). The system software should not place non-periodic schedule items into the periodic schedule. The least significant bits in a frame list pointer are used to key the host controller as to the type of object the pointer is referencing.

The least significant bit is the T-Bit (bit 0). When this bit is set, the host controller will never use the value of the frame list pointer as a physical memory pointer. The Typ field is used to indicate the exact type of data structure being referenced by this pointer. The value encodings for the Typ field are given in Table 24-37.

Table 24-37. Typ Field Encodings

Typ	Description
00	Isochronous Transfer Descriptor
01	Queue Head
10	Split Transaction Isochronous Transfer Descriptor
11	Frame Span Traversal Node.

24.8.2 Asynchronous List Queue Head Pointer

The Asynchronous Transfer List (based at the ASYNCLISTADDR register) is where all the control and bulk transfers are managed. Host controllers use this list only when it reaches the end of the periodic list, the periodic list is disabled, or the periodic list is empty.

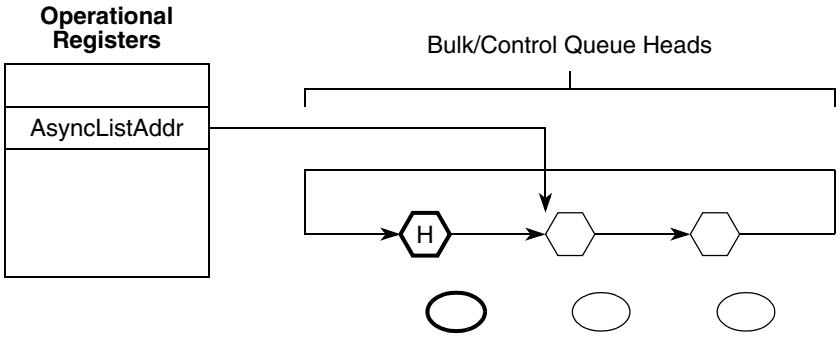


Figure 24-37. Asynchronous Schedule Organization

The Asynchronous list is a simple circular list of queue heads. The ASYNCLISTADDR register is simply pointer to the next queue head. This implements a pure round-robin service for all queue heads linked into the asynchronous list.

24.8.3 Isochronous (High-Speed) Transfer Descriptor (iTD)

The format of an isochronous transfer descriptor is illustrated in Figure 24-38. This structure is used only for high-speed isochronous endpoints. All other transfer types should use queue structures. Isochronous TDs must be aligned on a 32-byte boundary.

31–5	4–3	2–1	0	Offset
Next Link Pointer	00	Typ	T	0x00
Status[1] | Transaction 0 Length[1] | ioc | PG[2]	Transaction 0 Offset[2]			0x04
Status[1] | Transaction 1 Length[1] | ioc | PG[2]	Transaction 1 Offset[2]			0x08
Status[1] | Transaction 2 Length[1] | ioc | PG[2]	Transaction 2 Offset[2]			0x0C
Status[1] | Transaction 3 Length[1] | ioc | PG[2]	Transaction 3 Offset[2]			0x10
Status[1] | Transaction 4 Length[1] | ioc | PG[2]	Transaction 4 Offset[2]			0x14
Status[1] | Transaction 5 Length[1] | ioc | PG[2]	Transaction 5 Offset[2]			0x18
Status[1] | Transaction 6 Length[1] | ioc | PG[2]	Transaction 6 Offset[2]			0x1C
Status[1] | Transaction 7 Length[1] | ioc | PG[2]	Transaction 7 Offset[2]			0x20
Buffer Pointer (Page 0)	EndPt	R	Device Address	0x24
Buffer Pointer (Page 1)	I/O	Maximum Packet Size		0x28
Buffer Pointer (Page 2)	Reserved		Mult	0x2C
Buffer Pointer (Page 3)	Reserved			0x30
Buffer Pointer (Page 4)	Reserved			0x34
Buffer Pointer (Page 5)	Reserved			0x38
Buffer Pointer (Page 6)	Reserved			0x3C

Figure 24-38. Isochronous Transaction Descriptor (iTD)

1 Host controller read/write; all others read-only.

2 These fields may be modified by the host controller if the I/O field indicates an OUT.

24.8.3.1 Next Link Pointer

The first DWord of an iTD is a pointer to the next schedule data structure.

Table 24-38. Next Schedule Element Pointer

Bit	Name	Description
31–5	Link Pointer	These bits correspond to memory address signals [31–5], respectively. This field points to another Isochronous Transaction Descriptor (iTD/siTD) or Queue Head (QH).
4–3	–	Reserved. These bits are reserved and their value has no effect on operation. The software should initialize this field to zero.

Table 24-38. Next Schedule Element Pointer (continued)

Bit	Name	Description
2–1	Typ	This field indicates to the Host Controller whether the item referenced is an iTD, siTD or a QH. This allows the Host Controller to perform the proper type of processing on the item after it is fetched. Value encodings are: 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor 11 FSTN (frame span traversal node)
0	T	Terminate 1 Link Pointer field is not valid 0 Link Pointer field is valid

24.8.3.2 iTD Transaction Status and Control List

DWords 1 through 8 are eight slots of transaction control and status. Each transaction description includes:

- Status results field
- Transaction length (bytes to send for OUT transactions and bytes received for IN transactions).
- Buffer offset. The PG and Transaction *n* Offset fields are used with the buffer pointer list to construct the starting buffer address for the transaction.

The host controller uses the information in each transaction description plus the endpoint information contained in the first three DWords of the Buffer Page Pointer list, to execute a transaction on the USB.

Table 24-39. iTD Transaction Status and Control

Bit	Name	Description
31–28	Status	This field records the status of the transaction executed by the host controller for this slot. This field is a bit vector with the following encoding: 31 Active. Set by the software to enable the execution of an isochronous transaction by the Host Controller. When the transaction associated with this descriptor is completed, the Host Controller sets this bit to zero indicating that a transaction for this element should not be executed when it is next encountered in the schedule. 30 Data Buffer Error. Set by the Host Controller during status update to indicate that the Host Controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (underrun). If an overrun condition occurs, no action is necessary. 29 Babble Detected. Set by the Host Controller during status update when" babble" is detected during the transaction generated by this descriptor. 28 Transaction Error (XactErr). Set by the Host Controller during status update in the case where the host did not receive a valid response from the device (Time-out, CRC, Bad PID, etc.). This bit may only be set for isochronous IN transactions.
27–16	Transaction *n* Length	For an OUT, this field is the number of data bytes the host controller will send during the transaction. The host controller is not required to update this field to reflect the actual number of bytes transferred during the transfer. For an IN, the initial value of the endpoint to deliver. During the status update, the host controller writes back the field is the number of bytes the host expects the number of bytes successfully received. The value in this register is the actual byte count (for example, 0 zero length data, 1 one byte, 2 two bytes, etc.). The maximum value this field may contain is 0xC00 (3072).
15	ioc	Interrupt on complete. If this bit is set, it specifies that when this transaction completes, the Host Controller should issue an interrupt at the next interrupt threshold.

Table 24-39. iTD Transaction Status and Control (continued)

Bit	Name	Description
14–12	PG	These bits are set by the software to indicate which of the buffer page pointers the offset field in this slot should be concatenated to produce the starting memory address for this transaction. The valid range of values for this field is 0 to 6.
11–0	Transaction *n* Offset	This field is a value that is an offset, expressed in bytes, from the beginning of a buffer. This field is concatenated onto the buffer page pointer indicated in the adjacent PG field to produce the starting buffer address for this transaction.

24.8.3.3 iTD Buffer Page Pointer List (Plus)

DWords 9–15 of an isochronous transaction descriptor are nominally page pointers (4K aligned) to the data buffer for this transfer descriptor. This data structure requires the associated data buffer to be contiguous (relative to virtual memory), but allows the physical memory pages to be non-contiguous. Seven page pointers are provided to support the expression of eight isochronous transfers. The seven pointers allow for 3 (transactions) × 1024 (maximum packet size) × 8 (transaction records) = 24576 bytes to be moved with this data structure, regardless of the alignment offset of the first page.

Since each pointer is a 4K aligned page pointer, the least significant 12 bits in several of the page pointers are used for other purposes.

Table 24-40. Buffer Pointer Page 0 (Plus)

Bit	Name	Description
31–12	Buffer Pointer (Page 0)	This is a 4K aligned pointer to physical memory. Corresponds to memory address bits [31–12].
11–8	EndPt	This 4-bit field selects the particular endpoint number on the device serving as the data source or sink.
7	–	Reserved. Reserved for future use and should be initialized by he software to zero.
6–0	Device Address	This field selects the specific device serving as the data source or sink.

Table 24-41. iTD Buffer Pointer Page 1 (Plus)

Bit	Name	Description
31–12	Buffer Pointer (Page 1)	This is a 4K aligned pointer to physical memory. Corresponds to memory address bits [31–12].
11	I/O	Direction (I/O). This field encodes whether the high-speed transaction should use an IN or OUT PID. 0 Out 1 In
10–0	Maximum Packet Size	This directly corresponds to the maximum packet size of the associated endpoint (*wMaxPacketSize*). This field is used for high-bandwidth endpoints where more than one transaction is issued per transaction description (.for example, per micro-frame). This field is used with the *Multi* field to support high-bandwidth pipes. This field is also used for all IN transfers to detect packet babble. The software should not set a value larger than 1024 (400h). Any value larger yields undefined results.

Table 24-42. Buffer Pointer Page 2 (Plus)

Bit	Name	Description
31–12	Buffer Pointer (Page 2)	This is a 4K aligned pointer to physical memory. Corresponds to memory address bits [31–12].
11–2	–	Reserved. This bit reserved for future use and should be cleared.
1–0	Mult	This field is used to indicate to the host controller the number of transactions that should be executed per transaction description (for example, per micro-frame). The valid values are: 00 Reserved. A zero in this field yields undefined results. 01 One transaction to be issued for this endpoint per micro-frame 10 Two transactions to be issued for this endpoint per micro-frame 11 Three transactions to be issued for this endpoint per micro-frame

Table 24-43. Buffer Pointer Page 3–6

Bit	Name	Description
31–12	Buffer Pointer	This is a 4K aligned pointer to physical memory. Corresponds to memory address bits [31–12].
11–2	–	Reserved. These bits reserved for future use and should be cleared.

24.8.4 Split Transaction Isochronous Transfer Descriptor (siTD)

All Full-speed isochronous transfers through the internal transaction translator are managed using the siTD data structure. This data structure satisfies the operational requirements for managing the split transaction protocol.

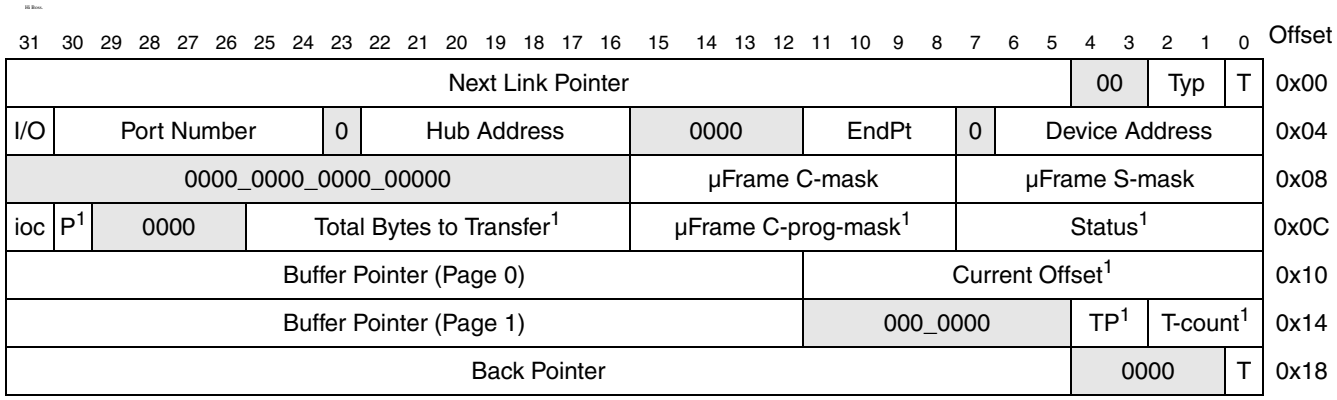


Figure 24-39. Split-Transaction Isochronous Transaction Descriptor (siTD)

1 Host controller read/write; all others read-only.

24.8.4.1 Next Link Pointer

DWord0 of a siTD is a pointer to the next schedule data structure.

Table 24-44. Next Link Pointer

Bit	Name	Description
31–5	Next Link Pointer	This field contains the address of the next data object to be processed in the periodic list and corresponds to memory address signals [31–5], respectively.
4–3	–	Reserved. These bits must be written as zeros.
2–1	Typ	This field indicates to the Host Controller whether the item referenced is an iTD/siTD or a QH. This allows the Host Controller to perform the proper type of processing on the item after it is fetched. Value encodings are: 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor 11 FSTN (frame span traversal node)
0	T	Terminate. 0 Link Pointer is valid 1 Link Pointer field is not valid

24.8.4.2 siTD Endpoint Capabilities/Characteristics

DWords 1 and 2 specify static information about the full-speed endpoint, the addressing of the parent Companion Controller, and micro-frame scheduling control.

Table 24-45. Endpoint and Transaction Translator Characteristics

Bit	Name	Description
31	I/O	Direction (I/O). This field encodes whether the full-speed transaction should be an IN or OUT. 0 Out 1 In
30–24	Port Number	This field is the port number of the recipient Transaction Translator.
23	–	Reserved. Bit reserved and should be cleared.
22–16	Hub Address	This field holds the device address of the Companion Controllers' hub.
15–12	–	Reserved. Field reserved and should be cleared.
11–8	EndPt	Endpoint Number. This 4-bit field selects the particular endpoint number on the device serving as the data source or sink.
7	–	Reserved. Bit is reserved for future use. It should be cleared.
6–0	Device Address	This field selects the specific device serving as the data source or sink.

Table 24-46. Micro-frame Schedule Control

Bit	Name	Description
31–16	–	Reserved. This field reserved for future use. It should be cleared.
15–8	µFrame C-mask	Split Completion Mask. This field (along with the Active and SplitX- state fields in the Status byte) is used to determine during which micro-frames the host controller should execute complete-split transactions. When the criteria for using this field is met, an all zeros value has undefined behavior. The host controller uses the value of the three low-order bits of the FRINDEX register to index into this bit field. If the FRINDEX register value indexes to a position where the µFrame C-Mask field is a one, then this siTD is a candidate for transaction execution. There may be more than one bit in this mask set.
7–0	µFrame S-mask	Split Start Mask. This field (along with the Active and SplitX-state fields in the Status byte) is used to determine during which micro-frames the host controller should execute start-split transactions. The host controller uses the value of the three low-order bits of the FRINDEX register to index into this bit field. If the FRINDEX register value indexes to a position where the µFrame S-mask field is a one, then this siTD is a candidate for transaction execution. An all zeros value in this field, in combination with existing periodic frame list has undefined results.

24.8.4.3 siTD Transfer State

DWords 3–6 are used to manage the state of the transfer.

Table 24-47. siTD Transfer Status and Control

Bit	Name	Description
31	ioc	Interrupt On Complete 0 Do not interrupt when transaction is complete. 1 Do interrupt when transaction is complete. When the host controller determines that the split transaction has completed, it will assert a hardware interrupt at the next interrupt threshold.
30	P	Page Select. Used to indicate which data page pointer should be concatenated with the CurrentOffset field to construct a data buffer pointer. 0 Selects Page 0 pointer 1 Selects Page 1 pointer The host controller is not required to write this field back when the siTD is retired (Active bit transitioned from a one to a zero).
29–26	–	Reserved. This field reserved for future use and should be cleared.
25–16	Total Bytes to Transfer	This field is initialized by the software to the total number of bytes expected in this transfer. Maximum value is 1023 (3FFh)
15–8	µFrame C-prog-mask	Split complete progress mask. This field is used by the host controller to record which split-completes have been executed.

Table 24-47. siTD Transfer Status and Control (continued)

Bit	Name	Description	
7–0	Status	This field records the status of the transaction executed by the host controller for this slot. This field is a bit vector with the following encoding:	
		Status Bit	**Definition**
		7	Active. Set by the software to enable the execution of an isochronous split transaction by the Host Controller.
		6	ERR. Set by the Host Controller when an ERR response is received from the Companion Controller.
		5	Data Buffer Error. Set by the Host Controller during status update to indicate that the Host Controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run). In the case of an under run, the Host Controller will transmit an incorrect CRC (thus invalidating the data at the endpoint). If an overrun condition occurs, no action is necessary.
		4	Babble Detected. Set by the Host Controller during status update when" babble" is detected during the transaction generated by this descriptor.
		3	Transaction Error (XactErr). Set by the Host Controller during status update in the case where the host did not receive a valid response from the device (Time-out, CRC, Bad PID, etc.). This bit will only be set for IN transactions.
		2	Missed Micro-Frame. The host controller detected that a host-induced hold- off caused the host controller to miss a required complete-split transaction.
		1	Split Transaction State (SplitXstate). The bit encodings are: 0 Do Start Split. This value directs the host controller to issue a Start split transaction to the endpoint when a match is encountered in the S-mask. 1 Do Complete Split. This value directs the host controller to issue a Complete split transaction to the endpoint when a match is encountered in the C-mask.
		0	Reserved. Bit reserved for future use and should be cleared.

24.8.4.4 siTD Buffer Pointer List (Plus)

DWords 4 and 5 are the data buffer page pointers for the transfer. This structure supports one physical page cross. The most significant 20 bits of each DWord in this section are the 4K (page) aligned buffer pointers. The least significant 12 bits of each DWord are used as additional transfer state.

Table 24-48. siTD Buffer Pointer Page 0 (Plus)

Bit	Name	Description
31–12	Buffer Pointer (Page 0)	Bits [31–12] is a 4K page-aligned, physical memory addresses. These bits correspond to physical address bits [31–12] respectively. The field P specifies the current active pointer
11–0	Current Offset	The 12 least significant bits of the Page 0 pointer is the current byte offset for the current page pointer (as selected with the page indicator bit (P field)). The host controller is not required to write this field back when the siTD is retired (Active bit transitioned from a one to a zero).

Table 24-49. siTD Buffer Pointer Page 1 (Plus)

Bit	Name	Description
31–12	Buffer Pointer (Page 1)	Bits [31–12] is a 4K page-aligned, physical memory addresses. These bits correspond to physical address bits [31–12] respectively. The field P specifies the current active pointer
11–5	–	Reserved.
4–3	TP	Transaction position. This field is used with T-count to determine whether to send all, first, middle, or last with each outbound transaction payload. The system software must initialize this field with the appropriate starting value. The host controller must correctly manage this state during the lifetime of the transfer. The bit encodings are: 00 All. The entire full-speed transaction data payload is in this transaction (that is, less than or equal to 188 bytes). 01 Begin. This is the first data payload for a full-speed transaction that is greater than 188 bytes. 10 Mid. This is the middle payload for a full-speed OUT transaction that is larger than 188 bytes. 11 End. This is the last payload for a full-speed OUT transaction that was larger than 188 bytes.
2–0	T-Count	Transaction count. The software initializes this field with the number of OUT start-splits this transfer requires. Any value larger than 6 is undefined.

24.8.4.5 siTD Back Link Pointer

DWord 6 of a siTD is simply another schedule link pointer. This pointer is always zero, or references a siTD. This pointer cannot reference any other schedule data structure.

Table 24-50. siTD Back Link Pointer

Bit	Name	Description
31–5	Back Pointer	This field is a physical memory pointer to a siTD.
4–1	–	Reserved. This field is reserved for future use. It should be cleared.
0	T	Terminate 0 siTD Back Pointer field is valid 1 siTD Back Pointer field is not valid

24.8.5 Queue Element Transfer Descriptor (qTD)

This data structure is only used with a queue head. This data structure is used for one or more USB transactions. This data structure is used to transfer up to 20480 (5 × 4096) bytes. The structure contains two structure pointers used for queue advancement, a DWord of transfer state, and a five-element array of data buffer pointers. This structure is 32 bytes (or one 32-byte cache line). This data structure must be physically contiguous.

The buffer associated with this transfer must be virtually contiguous. The buffer may start on any byte boundary. A separate buffer pointer list element must be used for each physical page in the buffer, regardless of whether the buffer is physically contiguous.

Host controller updates (host controller writes) to stand-alone qTDs only occur during transfer retirement. References in the following bit field definitions of updates to the qTD are to the qTD portion of a queue head.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Offset
Next qTD Pointer																											0000				T	0x00
Alternate Next qTD Pointer																											0000				T	0x04
dt[1]	Total Bytes to Transfer[1]															ioc	C_Page[1]			Cerr[1]		PID Code	Status[1]									0x08
Buffer Pointer (Page 0)																				Current Offset[1]												0x0C
Buffer Pointer (Page 1)																				0000_0000_0000												0x10
Buffer Pointer (Page 2)																				0000_0000_0000												0x14
Buffer Pointer (Page 3)																				0000_0000_0000												0x18
Buffer Pointer (Page 4)																				0000_0000_0000												0x1C

Figure 24-40. Queue Element Transfer Descriptor (qTD)

[1] Host controller read/write; all others read-only.

Queue Element Transfer Descriptors must be aligned on 32-byte boundaries.

24.8.5.1 Next qTD Pointer

The first DWord of an element transfer descriptor is a pointer to another transfer element descriptor.

Table 24-51. qTD Next Element Transfer Pointer (DWord 0)

Bit	Name	Description
31–5	Next qTD Pointer	This field contains the physical memory address of the next qTD to be processed. The field corresponds to memory address signals[31–5], respectively.
4–1	–	Reserved. These bits are reserved and their value has no effect on operation.
0	T	Terminate. This bit indicates to the Host Controller that there are no more valid entries in the queue. 0 Pointer is valid (points to a valid Transfer Element Descriptor). 1 Pointer is invalid.

24.8.5.2 Alternate Next qTD Pointer

The second DWord of a queue element transfer descriptor is used to support hardware-only advance of the data stream to the next client buffer on short packet. To be more explicit the host controller will always use this pointer when the current qTD is retired due to short packet.

Table 24-52. qTD Alternate Next Element Transfer Pointer (DWord 1)

Bit	Name	Description
31–5	Alternate Next qTD Pointer	This field contains the physical memory address of the next qTD to be processed in the event that the current qTD execution encounters a short packet (for an IN transaction). The field corresponds to memory address signals [31–5], respectively.

Table 24-52. qTD Alternate Next Element Transfer Pointer (DWord 1) (continued)

Bit	Name	Description
4–1	–	Reserved. These bits are reserved and their value has no effect on operation.
0	T	Terminate. This bit indicates to the Host Controller that there are no more valid entries in the queue. 0 Pointer is valid (points to a valid Transfer Element Descriptor). 1 Pointer is invalid.

24.8.5.3 qTD Token

The third DWord of a queue element transfer descriptor contains most of the information the host controller requires to execute a USB transaction (the remaining endpoint-addressing information is specified in the queue head). Note that some of the field descriptions in Table 24-53 reference fields defined in the queue head. See Section 24.8.6, "Queue Head," for more information on these fields.

Table 24-53. qTD Token (DWord 2)

Bit	Name	Description
31	dt	Data Toggle. This is the data toggle sequence bit. The use of this bit depends on the setting of the Data Toggle Control bit in the queue head.
30–16	Total Bytes to Transfer	Total Bytes to Transfer. This field specifies the total number of bytes to be moved with this transfer descriptor. This field is decremented by the number of bytes actually moved during the transaction, only on the successful completion of the transaction. The maximum value the software may store in this field is 5 × 4K (0x5000). This is the maximum number of bytes 5 page pointers can access. If the value of this field is zero when the host controller fetches this transfer descriptor (and the active bit is set), the host controller executes a zero-length transaction and retires the transfer descriptor. It is not a requirement for OUT transfers that Total Bytes To Transfer be an even multiple of QH[Maximum Packet Length]. If the software builds such a transfer descriptor for an OUT transfer, the last transaction will always be less than QH[Maximum Packet Length]. Although it is possible to create a transfer up to 20K this assumes the page is 0. When the offset cannot be predetermined, crossing past the 5th page can be guaranteed by limiting the total bytes to 16K. Therefore, the maximum recommended transfer is 16K(0x4000).
15	ioc	Interrupt On Complete. If this bit is set, it specifies that when this qTD is completed, the Host Controller should issue an interrupt at the next interrupt threshold.
14–12	C_Page	Current Page. This field is used as an index into the qTD buffer pointer list. Valid values are in the range 0x0 to 0x4. The host controller is not required to write this field back when the qTD is retired.

Table 24-53. qTD Token (DWord 2) (continued)

<table>
<tr><th>Bit</th><th>Name</th><th colspan="2">Description</th></tr>
<tr><td>11–10</td><td>Cerr</td><td colspan="2">Error Counter. This field is a 2-bit down counter that keeps track of the number of consecutive errors detected while executing this qTD. If this field is programmed with a non-zero value during set-up, the host controller decrements the count and writes it back to the qTD if the transaction fails. If the counter counts from one to zero, the host controller marks the qTD inactive, sets the Halted bit to a one, and error status bit for the error that caused Cerr to decrement to zero. An interrupt will be generated if the USB Error Interrupt Enable bit in the USBINTR register is set. If the host controller driver (HCD) software programs this field to zero during set-up, the host controller will not count errors for this qTD and there will be no limit on the retries of this qTD.
Note: Write-backs of intermediate execution state are to the queue head overlay area, not the qTD.</td></tr>
<tr><td></td><td></td><td>Error</td><td>Decrement Counter</td></tr>
<tr><td></td><td></td><td>Transaction Error</td><td>Yes</td></tr>
<tr><td></td><td></td><td>Data Buffer Error</td><td>No. Data buffer errors are host problems. They don't count against the device's retries.
Note: The software must not program Cerr to a value of zero when the EPS field is programmed with a value indicating a Full- or Low-speed device. This combination could result in undefined behavior.</td></tr>
<tr><td></td><td></td><td>Stalled</td><td>No. Detection of Babble or Stall automatically halts the queue head. Thus, count is not decremented</td></tr>
<tr><td></td><td></td><td>Babble Detected</td><td>No. Detection of Babble or Stall automatically halts the queue head. Thus, count is not decremented</td></tr>
<tr><td></td><td></td><td>No Error</td><td>No. If the EPS field indicates a HS device or the queue head is in the Asynchronous Schedule (and PIDCode indicates an IN or OUT) and a bus transaction completes and the host controller does not detect a transaction error, then the host controller should reset Cerr to extend the total number of errors for this transaction. For example, Cerr should be reset with maximum value (0b11) on each successful completion of a transaction. The host controller must never reset this field if the value at the start of the transaction is 0b00.</td></tr>
<tr><td>9–8</td><td>PID Code</td><td colspan="2">This field is an encoding of the token, which should be used for transactions associated with this transfer descriptor. Encodings are:
00 OUT Token generates token (E1H)
01 IN Token generates token (69H)
10 SETUP Token generates token (2DH) (undefined if endpoint is an Interrupt transfer type, for example. µFrame S-mask field in the queue head is non-zero.)
11 Reserved.</td></tr>
</table>

Table 24-53. qTD Token (DWord 2) (continued)

Bit	Name	Description	
7–0	Status	This field is used by the host controller to communicate individual command execution states back to the host controller driver (HCD) software. This field contains the status of the last transaction performed on this qTD. The bit encodings are:	
		Bit	**Status Field Description**
		7	Active. Set by the software to enable the execution of transactions by the host controller.
		6	Halted. Set by the host controller during status updates to indicate that a serious error has occurred at the device/endpoint addressed by this qTD. This can be caused by babble, the error counter counting down to zero, or reception of the STALL handshake from the device during a transaction. Any time that a transaction results in the Halted bit being set, the Active bit is also cleared.
		5	Data Buffer Error. Set by the host controller during status update to indicate that the host controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run). If an overrun condition occurs, the host controller will force a time-out condition on the USB, invalidating the transaction at the source. If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.
		4	Babble Detected. Set by the host controller during status update when babble is detected during the transaction. In addition to setting this bit, the host controller also sets the Halted bit to a one. Since babble is considered a fatal error for the transfer, setting the Halted bit to a one insures that no more transactions occur because of this descriptor.
		3	Transaction Error (XactErr). Set by the host controller during status update in the case where the host did not receive a valid response from the device (time-out, CRC, bad PID). If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.
		2	Missed Micro-Frame. This bit is ignored unless the QH[EPS] field indicates a full- or low-speed endpoint and the queue head is in the periodic list. This bit is set when the host controller detected that a host-induced hold-off caused the host controller to miss a required complete-split transaction. If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.
		1	Split Transaction State (SplitXstate). This bit is ignored by the host controller unless the QH[EPS] field indicates a full- or low-speed endpoint. When a full- or low-speed device, the host controller uses this bit to track the state of the split- transaction. The functional requirements of the host controller for managing this state bit and the split transaction protocol depends on whether the endpoint is in the periodic or asynchronous schedule. The bit encodings are: 0 Do Start Split. This value directs the host controller to issue a Start split transaction to the endpoint. 1 Do Complete Split. This value directs the host controller to issue a Complete split transaction to the endpoint.
		0	Ping State (P)/ERR. If the QH[EPS] field indicates a high-speed device and the PID Code indicates an OUT endpoint, then this is the state bit for the Ping protocol. The bit encodings are: 0 Do OUT. This value directs the host controller to issue an OUT PID to the endpoint. 1 Do Ping. This value directs the host controller to issue a PING PID to the endpoint. If the QH[EPS] field does not indicate a high-speed device, then this field is used as an error indicator bit. It is set by the host controller whenever a periodic split-transaction receives an ERR handshake.

24.8.5.4 qTD Buffer Page Pointer List

The last five DWords of a queue element transfer descriptor is an array of physical memory address pointers. These pointers reference the individual pages of a data buffer.

The system software initializes Current Offset field to the starting offset into the current page, where current page is selected via the value in the C_Page field.

Table 24-54. qTD Buffer Pointer

Bit	Name	Description
31–12	Buffer Pointer (page *n*)	Each element in the list is a 4K page aligned physical memory address. The lower 12 bits in each pointer are reserved (except for the first one), as each memory pointer must reference the start of a 4K page. The field C_Page specifies the current active pointer. When the transfer element descriptor is fetched, the starting buffer address is selected using C_Page (similar to an array index to select an array element). If a transaction spans a 4K buffer boundary, the host controller must detect the page-span boundary in the data stream, increment C_Page and advance to the next buffer pointer in the list, and conclude the transaction via the new buffer pointer.
11–0	Current Offset (Page 0)/ – (Pages 1-4)	This field is reserved in all pointers except the first one (that is, Page 0). The host controller should ignore all reserved bits. For the page 0 current offset interpretation, this field is the byte offset into the current page (as selected by C_Page). The host controller is not required to write this field back when the qTD is retired. The software should ensure the reserved fields are initialized to zeros.

24.8.6 Queue Head

Figure 24-41 shows the 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
Queue Head Horizontal Link Pointer		00 Typ T	0x00
RL C Maximum Packet Length H dtc EPS	EndPt	I Device Address	0x04[1]
Mult Port Number Hub Addr	µFrame C-mask	µFrame S-mask	0x08[1]
Current qTD Pointer[2]		00000	0x0C
Next qTD Pointer[2]		0000 T[2]	0x10[3]
Alternate Next qTD Pointer[2]		NakCnt[2] T[2]	0x14[3,4]
dt[1] Total Bytes to Transfer[2] ioc[2] C_Page[2]	Cerr[2] PID Code[2]	Status[2]	0x18[3,4]
Buffer Pointer (Page 0)[2]	Current Offset[2]		0x1C[3,4]
Buffer Pointer (Page 1)[2]	0000 C-prog-mask[2]		0x20[3,4]
Buffer Pointer (Page 2)[2]	S-bytes[2]	FrameTag[2]	0x24[3,4]
Buffer Pointer (Page 3)[2]	0000_0000_0000		0x28[3]
Buffer Pointer (Page 4)[2]	0000_0000_0000		0x2C[3]

Figure 24-41. Queue Head Layout

1 Offsets 0x04 through 0x0B contain the static endpoint state.
2 Host controller read/write; all others read-only.
3 Offsets 0x10 through 0x2F contain the transfer overlay.
4 Offsets 0x14 through 0x27 contain the transfer results.

24.8.6.1 Queue Head Horizontal Link Pointer

The first DWord of a Queue Head contains a link pointer to the next data object to be processed after any required processing in this queue has been completed, as well as the control bits defined below.

This pointer may reference a queue head or one of the isochronous transfer descriptors. It must not reference a queue element transfer descriptor.

Table 24-55. Queue Head DWord 0

Bit	Name	Description
31–5	QHLP	Queue Head Horizontal Link Pointer. This field contains the address of the next data object to be processed in the horizontal list and corresponds to memory address signals [31–5], respectively.
4–3	–	Reserved. These bits must be written as zeros.
2–1	Typ	This field indicates to the hardware whether the item referenced by the link pointer is an iTD, siTD or a QH. This allows the host controller to perform the proper type of processing on the item after it is fetched. 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor) 11 FSTN (frame span traversal node)
0	T	Terminate. 1 Last QH (pointer is invalid). 0 Pointer is valid. If the queue head is in the context of the periodic list, a one bit in this field indicates to the host controller that this is the end of the periodic list. This bit is ignored by the host controller when the queue head is in the Asynchronous schedule. The software must ensure that queue heads reachable by the host controller always have valid horizontal link pointers.

24.8.6.2 Endpoint Capabilities/Characteristics

The second and third DWords of a Queue Head specifies static information about the endpoint. This information does not change over the lifetime of the endpoint. There are three types of information in this region:

- Endpoint Characteristics. These are the USB endpoint characteristics including addressing, maximum packet size, and endpoint speed.
- Endpoint Capabilities. These are adjustable parameters of the endpoint. They effect how the endpoint data stream is managed by the host controller.
- Split Transaction Characteristics. This data structure is used to manage full- and low-speed data streams for bulk, control, and interrupt via split transactions to USB2.0 Hub Transaction Translator. There are additional fields used for addressing the hub and scheduling the protocol transactions (for periodic).

The host controller must not modify the bits in this region.

Table 24-56. Endpoint Characteristics: Queue Head DWord 1

Bit	Name	Description
31–28	RL	Nak count reload. This field contains a value, which is used by the host controller to reload the Nak Counter field.
27	C	Control endpoint flag. If the QH[EPS] field indicates the endpoint is not a high-speed device, and the endpoint is a control endpoint, then the software must set this bit to a one. Otherwise, it should always set this bit to a zero.
26–16	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	H	Head of reclamation list flag. This bit is set by the system software to mark a queue head as being the head of the reclamation list.
14	dtc	Data Toggle Control (DTC). This bit specifies where the host controller should get the initial data toggle on an overlay transition. 0 Ignore DT bit from incoming qTD. Host controller preserves DT bit in the queue head. 1 Initial data toggle comes from incoming qTD DT bit. Host controller replaces DT bit in the queue head from the DT bit in the qTD.
13–12	EPS	Endpoint Speed. This is the speed of the associated endpoint. 00 Full-Speed (12Mbs) 01 Low-Speed (1.5Mbs) 10 High-Speed (480 Mb/s) 11 Reserved. This field must not be modified by the host controller.
11–8	EndPt	Endpoint Number. This 4-bit field selects the particular endpoint number on the device serving as the data source or sink.
7	I	Inactivate on next transaction. This bit is used by the system software to request that the host controller set the Active bit to zero. This field is only valid when the queue head is in the Periodic Schedule and the EPS field indicates a Full or Low-speed endpoint. Setting this bit to a one when the queue head is in the Asynchronous Schedule or the EPS field indicates a high-speed device yields undefined results.
6–0	Device Address	This field selects the specific device serving as the data source or sink.

Table 24-57. Endpoint Capabilities: Queue Head DWord 2

Bit	Name	Description
31–30	Mult	High-Bandwidth Pipe Multiplier. This field is a multiplier used to key the host controller as the number of successive packets the host controller may submit to the endpoint in the current execution. The host controller makes the simplifying assumption that the software properly initializes this field (regardless of location of queue head in the schedules or other run time parameters). 00 Reserved. A zero in this field yields undefined results. 01 One transaction to be issued for this endpoint per micro-frame 10 Two transactions to be issued for this endpoint per micro-frame 11 Three transactions to be issued for this endpoint per micro-frame
29–23	Port Number	This field is ignored by the host controller unless the EPS field indicates a full- or low-speed device. The value is the port number identifier on the USB 2.0 hub (for hub at device address Hub Addr below), below which the full- or low-speed device associated with this endpoint is attached. This information is used in the split-transaction protocol.

Table 24-57. Endpoint Capabilities: Queue Head DWord 2 (continued)

Bit	Name	Description
22–16	Hub Addr	This field is ignored by the host controller unless the EPS field indicates a full-or low-speed device. The value is the USB device address of the USB 2.0 hub below which the full- or low-speed device associated with this endpoint is attached. This field is used in the split-transaction protocol.
15–8	µFrame C-mask	This field is ignored by the host controller unless the EPS field indicates this device is a low- or full-speed device and this queue head is in the periodic list. This field (along with the Active and SplitX-state fields) is used to determine during which micro-frames the host controller should execute a complete-split transaction. When the criteria for using this field are met, a zero value in this field has undefined behavior. This field is used by the host controller to match against the three low-order bits of the FRINDEX register. If the FRINDEX register bits decode to a position where the µFrame C- mask field is a one, then this queue head is a candidate for transaction execution. There may be more than one bit in this mask set.
7–0	µFrame S-mask	Interrupt Schedule Mask. This field is used for all endpoint speeds. The software should set this field to a zero when the queue head is on the asynchronous schedule. A non-zero value in this field indicates an interrupt endpoint. The host controller uses the value of the three low-order bits of the FRINDEX register as an index into a bit position in this bit vector. If the µFrame S-mask field has a one at the indexed bit position then this queue head is a candidate for transaction execution. If the EPS field indicates the endpoint is a high-speed endpoint, then the transaction executed is determined by the PID_Code field contained in the execution area. This field is also used to support split transaction types: Interrupt (IN/OUT). This condition is true when this field is non-zero and the EPS field indicates this is either a full- or low-speed device. A zero value in this field, in combination with existing in the periodic frame list has undefined results.

24.8.6.3 Transfer Overlay

The nine DWords in this area represent a transaction working space for the host controller. The general operational model is that the host controller can detect whether the overlay area contains a description of an active transfer. If it does not contain an active transfer, then it follows the Queue Head Horizontal Link Pointer to the next queue head. The host controller will never follow the Next Transfer Queue Element or Alternate Queue Element pointers unless it is actively attempting to advance the queue. For the duration of the transfer, the host controller keeps the incremental status of the transfer in the overlay area. When the transfer is complete, the results are written back to the original queue element.

The DWord3 of a Queue Head contains a pointer to the source qTD currently associated with the overlay. The host controller uses this pointer to write back the overlay area into the source qTD after the transfer is complete.

Table 24-58. Current qTD Link Pointer

Bit	Name	Description
31–5	Current qTD Pointer	Current Element Transaction Descriptor Link Pointer. This field contains the address Of the current transaction being processed in this queue and corresponds to memory address signals [31–5], respectively.
4–0	–	Reserved. These bits are ignored by the host controller when using the value as an address to write data. The actual value may vary depending on the usage.

The DWords 4–11 of a queue head are the transaction overlay area. This area has the same base structure as a Queue Element Transfer Descriptor. The queue head utilizes the reserved fields of the page pointers to implement tracking the state of split transactions.

This area is characterized as an overlay because when the queue is advanced to the next queue element, the source queue element is merged onto this area. This area serves an execution cache for the transfer.

Table 24-59. Host-Controller Rules for Bits in Overlay (DWords 5, 6, 8 and 9)

DWord	QH Offset	Bit	Name	Description
5	0x14	4–1	NakCnt	Nak counter—RW. This field is a counter the host controller decrements whenever a transaction for the endpoint associated with this queue head results in a Nak or Nyet response. This counter is reloaded from RL before a transaction is executed during the first pass of the reclamation list (relative to an Asynchronous List Restart condition). It is also loaded from RL during an overlay.
6	0x18	31	dt	Data toggle. The Data Toggle Control controls whether the host controller preserves this bit when an overlay operation is performed.
6	0x18	15	ioc	Interrupt on complete. The ioc control bit is always inherited from the source qTD when the overlay operation is performed.
6	0x18	11–10	Cerr	Error counter. This two-bit field is copied from the qTD during the overlay and written back during queue advancement.
6	0x18	0	Status[0]	Ping state (P)/ERR. If the EPS field indicates a high-speed endpoint, then this field should be preserved during the overlay operation.
8	0x20	7–0	C-prog-mask	Split-transaction complete-split progress. This field is initialized to zero during any overlay. This field is used to track the progress of an interrupt split-transaction.
9	0x24	11–5	S-bytes	The software must ensure that the S-bytes field in a qTD is zero before activating the qTD. This field is used to keep track of the number of bytes sent or received during an IN or OUT split transaction.
9	0x24	4–0	FrameTag	Split-transaction frame tag. This field is initialized to zero during any overlay. This field is used to track the progress of an interrupt split-transaction.

24.8.7 Periodic Frame Span Traversal Node (FSTN)

This data structure is to be used only for managing Full- and Low-speed transactions that span a Host-frame boundary. The software must not use an FSTN in the Asynchronous Schedule. An FSTN in the Asynchronous schedule results in undefined behavior. The software must not use the FSTN feature with a host controller whose HCIVERSION register indicates a revision implementation below 0x0096. Note that FSTNs were not defined for EHCI implementations before Revision 0.96 of the EHCI Specification and their use may yield undefined results.

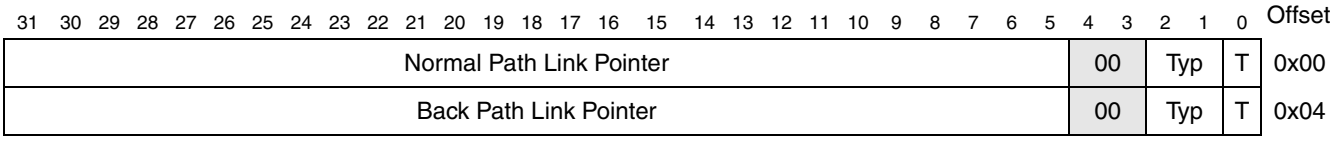


Figure 24-42. Frame Span Traversal Node Structure

24.8.7.1 FTSN Normal Path Pointer

The first DWord of an FSTN contains a link pointer to the next schedule object. This object can be of any valid periodic schedule data type.

Table 24-60. FTSN Normal Path Pointer

Bit	Name	Description
31–5	NPLP	Normal Path Link Pointer. This field contains the address of the next data object to be processed in the periodic list and corresponds to memory address signals [31–5], respectively.
4–3	–	Reserved. These bits must be written as 0s.
2–1	Typ	This field indicates to the host controller whether the item referenced is a iTD/siTD, a QH or an FSTN. This allows the host controller to perform the proper type of processing on the item after it is fetched. 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor) 11 FSTN (Frame Span Traversal Node)
0	T	Terminate. 0 Link Pointer is valid. 1 Link Pointer field is not valid.

24.8.7.2 FSTN Back Path Link Pointer

The second DWord of an FTSN node contains a link pointer to a queue head. If the T-bit in this pointer is a zero, then this FSTN is a Save-Place indicator. Its Typ field must be set by the software to indicate the target data structure is a queue head. If the T-bit in this pointer is set, then this FSTN is the Restore indicator. When the T-bit is a one, the host controller ignores the Typ field.

Table 24-61. FSTN Back Path Link Pointer

Bit	Name	Description
31–5	BPLP	Back Path Link Pointer. This field contains the address of a Queue Head. This field corresponds to memory address signals [31–5], respectively.
4–3	–	Reserved. These bits must be written as 0s.
2–1	Typ	The software must ensure this field is set to indicate the target data structure is a Queue Head (01). Any other value in this field yields undefined results.
0	T	Terminate. 0 Link Pointer is valid (that is, the host controller may use bits [31–5] (in combination with the CTRLDSSEGMENT register if applicable) as a valid memory address). This value also indicates that this FSTN is a Save-Place indicator. 1 Link Pointer field is not valid (that is, the host controller must not use bits [31–5] (in combination with the CTRLDSSEGMENT register if applicable) as a valid memory address). This value also indicates that this FSTN is a Restore indicator.

24.9 Host Operations

The general operational model is for the USB modules in host mode is defined by the Enhanced Host Controller Interface (EHCI) Specification. The EHCI specification describes the register-level interface for a host controller for the USB Revision 2.0. It includes a description of the hardware/software interface

between the system software and the host controller hardware. Information concerning the initialization of the USB modules is included in the following section; however, the full details of the EHCI specification are beyond the scope of this document.

24.9.1 Host Controller Initialization

After initial power-on or HCReset (hardware or via HCReset bit in the USBCMD register), all of the operational registers will be at their default values, as illustrated in Table 24-62. After a hardware reset, only the operational registers not contained in the Auxiliary power well will be at their default values.

Table 24-62. Default Values of Operational Register Space

Operational Register	Default Value (after Reset)
USBCMD	0x0008_0000
USBSTS	0x0000_1000
USBINTR	0x0000_0000
FRINDEX	0x0000_0000
CTRLDSSEGMENT	0x0000_0000
PERIODICLISTBASE	Undefined
ASYNCLISTADDR	Undefined
CONFIGFLAG	0x0000_00001
PORTSC	0x1c00_0004

In order to initialize the host controller, the software should perform the following steps:

1. Optionally set Streaming Disable in the USBMODE register.
2. Optionally modify the BURSTSIZE register.
3. Program the PTS field of the PORTSCx register.
4. Set the USB_EN bit in the CONTROL register.
5. Program the CTRLDSSEGMENT register with 4-Gigabyte segment where all of the interface data structures are allocated.
6. Write the appropriate value to the USBINTR register to enable the appropriate interrupts.
7. Write the base address of the Periodic Frame List to the PERIODICLIST BASE register. If there are no work items in the periodic schedule, all elements of the Periodic Frame List should have their *T-Bits* set.
8. Write the USBCMD register to set the desired interrupt threshold, frame list size (if applicable) and turn the controller on via setting the *Run/Stop* bit.

At this point, the host controller is up and running and the port registers begin reporting device connects. The system software can enumerate a port through the reset process (where the port is in the enabled state). At this point, the port is active with SOFs occurring down the enabled port enabled High-speed ports, but the schedules have not yet been enabled. The EHCI host controller will not transmit SOFs to enabled Full- or Low-speed ports.

In order to communicate with devices via the asynchronous schedule, the system software must write the ASYNDLISTADDR register with the address of a control or bulk queue head. The software must then enable the asynchronous schedule by writing a one to the *Asynchronous Schedule Enable* bit in the USBCMD register. In order to communicate with devices via the periodic schedule, the system software must enable the periodic schedule by writing a one to the *Periodic Schedule Enable* bit in the USBCMD register. Note that the schedules can be turned on before the first port is reset (and enabled).

Any time the USBCMD register is written, the system software must ensure the appropriate bits are preserved, depending on the intended operation.

24.9.2 Power Port

The Port Power Control (PPC) bit in the HCSPARAMS register indicates whether the USB 2.0 host controller has port power control. When the PPC bit is a one, then the host controller supports port power switches. Each available switch has an output enable. PPE is controlled based on the state of the combination bits PPC bit, EHCI Configured (CF)-bit and individual Port Power (PP) bits.

24.9.3 Reporting Over-Current

Host ports by definition are power providers on USB. Whether the ports are considered high- or low-powered is a platform implementation issue. Each EHCI PORTSC register has an over-current status and over-current change bit. The functionality of these bits is specified in the USB Specification Revision 2.0.

In this implementation, however, over-current is not reported to the USB core. Therefore the bits: *Over-current Active* and *Over-current Change* in the PORTSC register will be static. The over-current detection and limiting logic resides outside the MCF5251. The USB software stack is responsible for monitoring the Over-current condition on the external device.

24.9.4 Suspend/Resume

The host controller provides an equivalent suspend and resume model as that defined for individual ports in a USB 2.0 hub. Control mechanisms are provided to allow the system software to suspend and resume individual ports. The mechanisms allow the individual ports to be resumed completely via software initiation. Other control mechanisms are provided to parameterize the host controller's response (or sensitivity) to external resume events. In this discussion, host-initiated, or software-initiated resumes are called Resume Events/Actions; bus-initiated resume events are called wake-up events. The classes of wakeup events are:

- Remote-wakeup enabled device asserts resume signaling. In similar kind to USB 2.0 hubs, when in host mode the host controller responds to explicit device resume signaling and wake up the system (if necessary).
- Port connect and disconnect. Sensitivity to these events can be turned on or off by using the port control bits in the PORTSC register. An Over-current event will not wake the USB core.

Selective suspend is a feature supported by the PORTSC register. It is used to place specific ports into a suspend mode. This feature is used as a functional component for implementing the appropriate power

management policy implemented in a particular operating system. When the system software intends to suspend the bus, it should suspend the enabled port, then shut off the controller by setting the Run/Stop bit in the USBCMD register to a zero.

When a wake event occurs the system will resume operation, and the system software must set the Run/Stop bit to a one and resume the suspended port.

24.9.4.1 Port Suspend/Resume

The system software places the USB into suspend mode by writing a one into the appropriate PORTSC Suspend bit. The software must only set the Suspend bit when the port is in the enabled state (Port Enabled bit is a one).

The host controller may evaluate the Suspend bit immediately or wait until a micro-frame or frame boundary occurs. If evaluated immediately, the port is not suspended until the current transaction (if one is executing) completes. Therefore, there may be several micro-frames of activity on the port until the host controller evaluates the Suspend bit. The host controller must evaluate the Suspend bit at least every frame boundary.

The system software can initiate a resume on the suspended port by writing a one to the Force Port Resume bit. The software should not attempt to resume a port unless the port reports that it is in the suspended state. If the system software sets the Force Port Resume bit when the port is not in the suspended state, the resulting behavior is undefined. In order to assure proper USB device operation, the software must wait for at least 10 milliseconds after a port indicates that it is suspended (Suspend bit is a one) before initiating a port resume via the Force Port Resume bit. When Force Port Resume bit is set, the host controller sends resume signaling down the port. The system software times the duration of the resume (nominally 20 milliseconds) then clears the Force Port Resume bit. When the host controller receives the write to transition Force Port Resume to zero, it completes the resume sequence as defined in the USB specification, and clears both the Force Port Resume and Suspend bits. Software-initiated port resumes do not affect the Port Change Detect bit in the USBSTS register nor do they cause an interrupt if the Port Change Interrupt Enable bit in the USBINTR register is a one. When a wake event occurs on a suspended port, the resume signaling is detected by the port and the resume is reflected downstream within 100 µsec. The port's Force Port Resume bit is set and the Port Change Detect bit in the USBSTS register is set. If the Port Change Interrupt Enable bit in the USBINTR register is a one the host controller issues a hardware interrupt.

The system software observes the resume event on the port, delays a port resume time (nominally 20 milliseconds), then terminates the resume sequence by clearing the Force Port Resume bit in the port. The host controller receives the write of zero to Force Port Resume, terminates the resume sequence and clears the Force Port Resume and Suspend port bits. The software can determine that the port is enabled (not suspended) by sampling the PORTSC register and observing that the Suspend and Force Port Resume bits are zero. The software must ensure that the host controller is running (that is, HCHalted bit in the USBSTS register is a zero), before terminating a resume by clearing the port's Force Port Resume bit. If HCHalted is a one when Force Port Resume is cleared, then SOFs will not occur down the enabled port and the device will return to suspend mode in a maximum of 10 milliseconds.

Table 24-63 summarizes the wake-up events. Whenever a resume event is detected, the Port Change Detect bit in the USBSTS register is set. If the Port Change Interrupt Enable bit is a one in the USBINTR register,

the host controller also generates an interrupt on the resume event. The software acknowledges the resume event interrupt by clearing the Port Change Detect status bit in the USBSTS register.

Table 24-63. Behavior During Wake-up Events

Port Status and Signaling Type	Signaled Port Response	Device State	
		D0	not D0
Port disabled, resume K-State received	No Effect	N/A	N/A
Port suspended, Resume K-State received	Resume reflected downstream on signaled port. Force Port Resume status bit in PORTSC register is set. Port Change Detect bit in USBSTS register is set.	[1], [2]	[2]
Port is enabled, disabled or suspended, and the port's WKDSCNNT_E bit is set. A disconnect is detected.	Depending in the initial port state, the PORTSC Connect and Enable status bits are cleared, and the Connect Change status bit is set. Port Change Detect bit in the USBSTS register is set.	[1], [2]	[2]
Port is enabled, disabled or suspended, and the port's WKDSCNNT_E bit is cleared. A disconnect is detected.	Depending on the initial port state, the PORTSC Connect and Enable status bits are cleared, and the Connect Change status bit is set. Port Change Detect bit in the USBSTS register is set.	[1], [3]	[3]
Port is not connected and the port's WKCNNT_E bit is a one. A connect is detected.	PORTSC Connect Status and Connect Status Change bits are set. Port Change Detect bit in the USBSTS register is set.	[1], [2]	[2]
Port is not connected and the port's WKCNNT_E bit is a zero. A connect is detected.	PORTSC Connect Status and Connect Status Change bits are set. Port Change Detect bit in the USBSTS register is set.	[1], [3]	[3]
Port is connected and the port's WKOC_E bit is a one. An over-current condition occurs.	PORTSC Over-current Active, Over-current Change bits are set. If Port Enable/Disable bit is a one, it is cleared. Port Change Detect bit in the USBSTS register is set	[1], [2]	[2]
Port is connected and the port's WKOC_E bit is a zero. An over-current condition occurs.	PORTSC Over-current Active, Over-current Change bits are set. If Port Enable/Disable bit is a one, it is cleared. Port Change Detect bit in the USBSTS register is set.	[1], [3]	[3]

[1] Hardware interrupt issued if Port Change Interrupt Enable bit in the USBINTR register is set.

[2] PME# asserted if enabled (Note: PME Status must always be set).

[3] PME# not asserted.

24.9.5 Schedule Traversal Rules

The host controller executes transactions for devices using a simple, shared-memory schedule. The schedule is comprised of a few data structures, organized into two distinct lists. The data structures are designed to provide the maximum flexibility required by USB and minimize memory traffic and hardware/software complexity.

The system software maintains two schedules for the host controller: a periodic schedule and an asynchronous schedule. The root of the periodic schedule is the PERIODICLISTBASE register. See Section 24.6.3.6, “Periodic Frame List Base Address Register (PERIODICLISTBASE),” for more information. The PERIODICLISTBASE register is the physical memory base address of the periodic frame list. The periodic frame list is an array of physical memory pointers. The objects referenced from the frame list must be valid schedule data structures as defined in Section 24.8, “Host Data Structures.” In

each micro-frame, if the periodic schedule is enabled (see) then the host controller must execute from the periodic schedule before executing from the asynchronous schedule. It will only execute from the asynchronous schedule after it encounters the end of the periodic schedule. The host controller traverses the periodic schedule by constructing an array offset reference from the PERIODICLISTBASE and the FRINDEX registers (see Figure 24-43). It fetches the element and begins traversing the graph of linked schedule data structures.

The end of the periodic schedule is identified by a next link pointer of a schedule data structure having its T-bit set. When the host controller encounters a T-Bit set during a horizontal traversal of the periodic list, it interprets this as an End-Of-Periodic-List mark. This causes the host controller to cease working on the periodic schedule and transitions immediately to traversing the asynchronous schedule. Once this transition is made, the host controller executes from the asynchronous schedule until the end of the micro-frame.

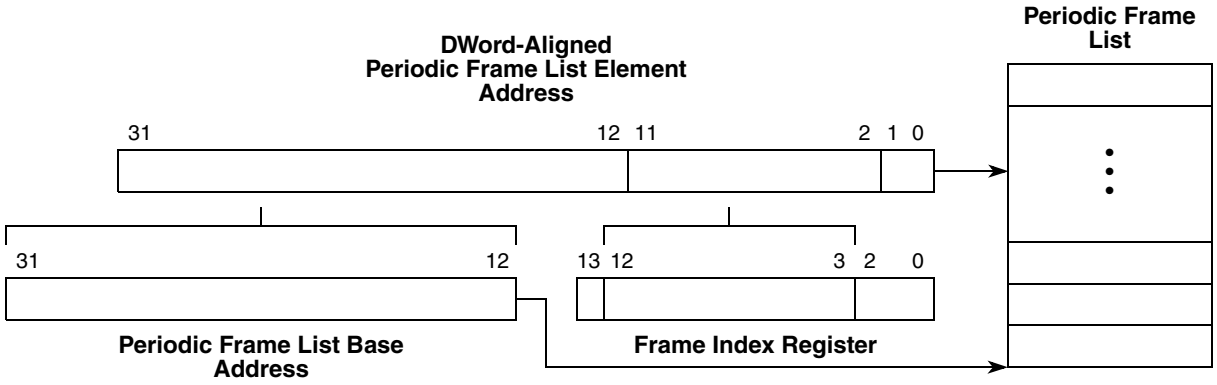


Figure 24-43. Derivation of Pointer into Frame List Array

When the host controller determines that it is time to execute from the asynchronous list, it uses the operational register ASYNCLISTADDR to access the asynchronous schedule, as shown in Figure 24-44.

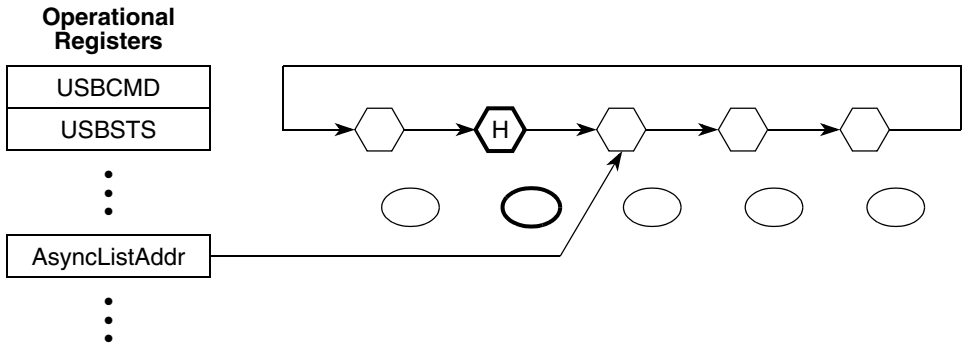


Figure 24-44. General Format of Asynchronous Schedule List

The ASYNCLISTADDR register contains a physical memory pointer to the next queue head. When the host controller makes a transition to executing the asynchronous schedule, it begins by reading the queue head referenced by the ASYNCLISTADDR register. The software must set queue head horizontal pointer T-bits to a zero for queue heads in the asynchronous schedule.

See Section 24.9.9, “Asynchronous Schedule” for complete operational details.

24.9.6 Periodic Schedule Frame Boundaries vs. Bus Frame Boundaries

The USB Specification Revision 2.0 requires that the frame boundaries (SOF frame number changes) of the high-speed bus and the full- and low-speed bus(es) below USB 2.0 hubs be strictly aligned. Super-imposed on this requirement is that USB 2.0 hubs manage full- and low-speed transactions via a micro-frame pipeline (see start- (SS) and complete- (CS) splits illustrated in Figure 24-45). A simple, direct projection of the frame boundary model into the host controller interface schedule architecture creates tension (complexity for both hardware and software) between the frame boundaries and the scheduling mechanisms required to service the full- and low-speed transaction translator periodic pipelines.

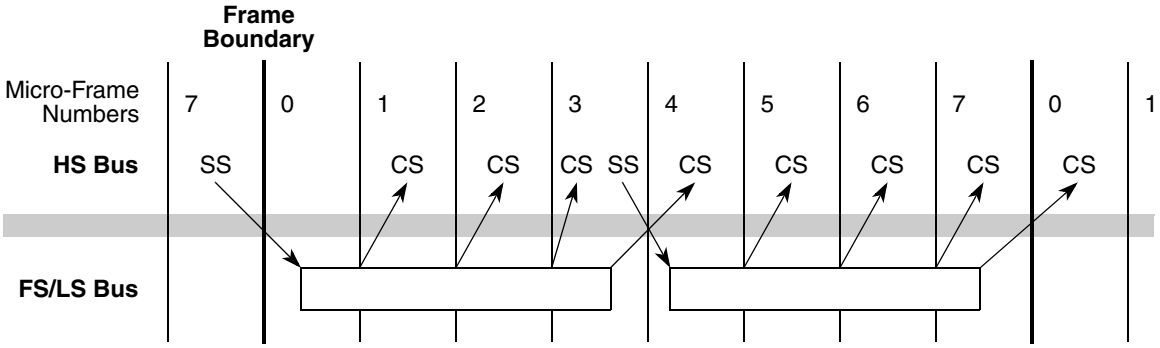


Figure 24-45. Frame Boundary Relationship between HS Bus and FS/LS Bus

The simple projection, as Figure 24-45 illustrates, introduces frame-boundary wrap conditions for scheduling on both the beginning and end of a frame. In order to reduce the complexity for hardware and software, the host controller is required to implement a one micro-frame phase shift for its view of frame boundaries. The phase shift eliminates the beginning of frame and frame-wrap scheduling boundary conditions.

The implementation of this phase shift requires that the host controller use one register value for accessing the periodic frame list and another value for the frame number value included in the SOF token. These two values are separate, but tightly coupled. The periodic frame list is accessed via the Frame List Index Register (FRINDEX). Bits FRINDEX[2:0], represent the micro-frame number. The SOF value is coupled to the value of FRINDEX[13:3]. Both FRINDEX[13:3] and the SOF value are incremented based on FRINDEX[2:0]. It is required that the SOF value be delayed from the FRINDEX value by one micro-frame. The one micro-frame delay yields a host controller periodic schedule and bus frame boundary relationship as illustrated in Figure 24-46. This adjustment allows the software to trivially schedule the periodic start and complete-split transactions for full-and low-speed periodic endpoints, using the natural alignment of the periodic schedule interface.

Figure 24-46 illustrates how periodic schedule data structures relate to schedule frame boundaries and bus frame boundaries. To aid the presentation, two terms are defined. The host controller's view of the 1-millisecond boundaries is called H-Frames. The high-speed bus's view of the 1-millisecond boundaries is called B-Frames.

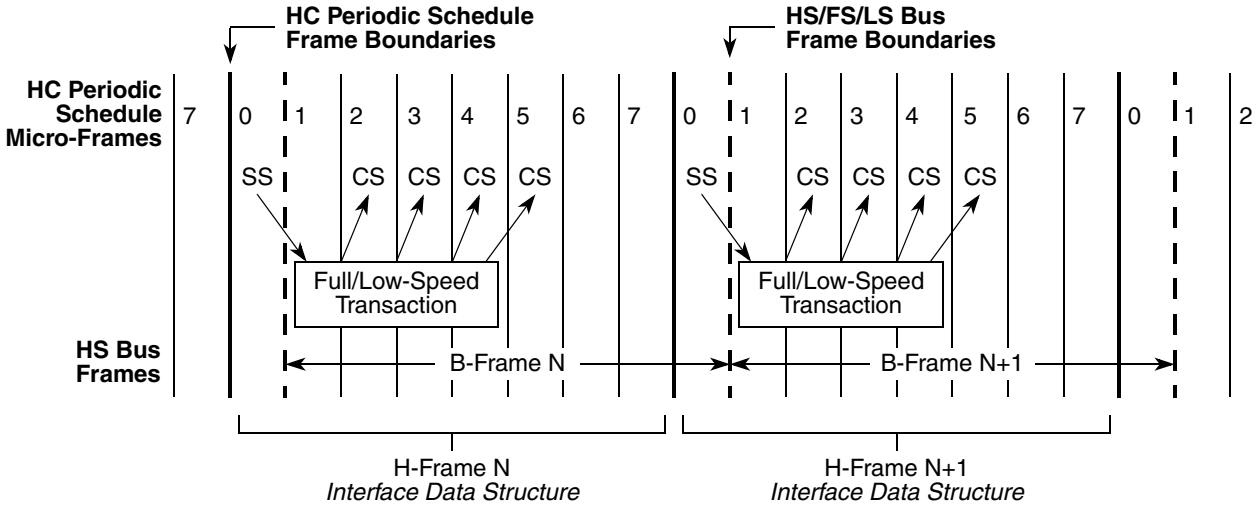


Figure 24-46. Relationship of Periodic Schedule Frame Boundaries to Bus Frame Boundaries

H-Frame boundaries for the host controller correspond to increments of FRINDEX[13:3]. Micro-frame numbers for the H-Frame are tracked by FRINDEX[2:0]. B-Frame boundaries are visible on the high-speed bus via changes in the SOF token's frame number. Micro-frame numbers on the high-speed bus are only derived from the SOF token's frame number (that is, the high-speed bus will see eight SOFs with the same frame number value). H-Frames and B-Frames have the fixed relationship (that is, B-Frames lag H-Frames by one micro-frame time) illustrated in Figure 24-46. The host controller's periodic schedule is naturally aligned to H-Frames. The software schedules transactions for full- and low-speed periodic endpoints relative the H-Frames. The result is these transactions execute on the high-speed bus at exactly the right time for the USB 2.0 hub periodic pipeline. As described in Section 24.6.3.4, “Frame Index Register (FRINDEX),” the SOF Value can be implemented as a shadow register (in this example, called SOFV), which lags the FRINDEX register bits [13:3] by one micro-frame count. Table 24-64 illustrates the required relationship between the value of FRINDEX and the value of SOFV. This lag behavior can be accomplished by incrementing FRINDEX[13:3] based on carry-out on the 7 to 0 increment of FRINDEX[2:0] and incrementing SOFV based on the transition of 0 to 1 of FRINDEX[2:0].

The software is allowed to write to FRINDEX. Section 24.6.3.4, “Frame Index Register (FRINDEX),” provides the requirements that the software should adhere when writing a new value in FRINDEX.

Table 24-64. Operation of FRINDEX and SOFV (SOF Value Register)

Current			Next		
FRINDEX[13:3]	SOFV	FRINDEX[2:0]	FRINDEX[13:3]	SOFV	FRINDEX[2:0]
N	N	111	N+1	N	000
N+1	N	000	N+1	N+1	001
N+1	N+1	001	N+1	N+1	010
N+1	N+1	010	N+1	N+1	011
N+1	N+1	011	N+1	N+1	100
N+1	N+1	100	N+1	N+1	101
N+1	N+1	101	N+1	N+1	110
N+1	N+1	110	N+1	N+1	111

24.9.7 Periodic Schedule

The periodic schedule traversal is enabled or disabled via the Periodic Schedule Enable bit in the USBCMD register. If the Periodic Schedule Enable bit is cleared, then the host controller simply does not try to access the periodic frame list via the PERIODICLISTBASE register. Likewise, when the Periodic Schedule Enable bit is a one, then the host controller does use the PERIODICLISTBASE register to traverse the periodic schedule. The host controller will not react to modifications to the Periodic Schedule Enable immediately. In order to eliminate conflicts with split transactions, the host controller evaluates the Periodic Schedule Enable bit only when FRINDEX[2:0] is zero. The system software must not disable the periodic schedule if the schedule contains an active split transaction work item that spans the 0b000 micro-frame. These work items must be removed from the schedule before the Periodic Schedule Enable bit is cleared. The Periodic Schedule Status bit in the USBSTS register indicates status of the periodic schedule. The system software enables (or disables) the periodic schedule by setting (or clearing) the Periodic Schedule Enable bit in the USBCMD register. The software then can poll the Periodic Schedule Status bit to determine when the periodic schedule has made the desired transition. The software must not modify the Periodic Schedule Enable bit unless the value of the Periodic Schedule Enable bit equals that of the Periodic Schedule Status bit.

The periodic schedule is used to manage all isochronous and interrupt transfer streams. The base of the periodic schedule is the periodic frame list. The software links schedule data structures to the periodic frame list to produce a graph of scheduled data structures. The graph represents an appropriate sequence of transactions on the USB. Figure 24-47 illustrates isochronous transfers (using iTDs and siTDs) with a period of one are linked directly to the periodic frame list. Interrupt transfers (are managed with queue heads) and isochronous streams with periods other than one are linked following the period-one iTD/siTDs. Interrupt queue heads are linked into the frame list ordered by poll rate. Longer poll rates are linked first (for example, closest to the periodic frame list), followed by shorter poll rates, with queue heads with a poll rate of one, on the very end.

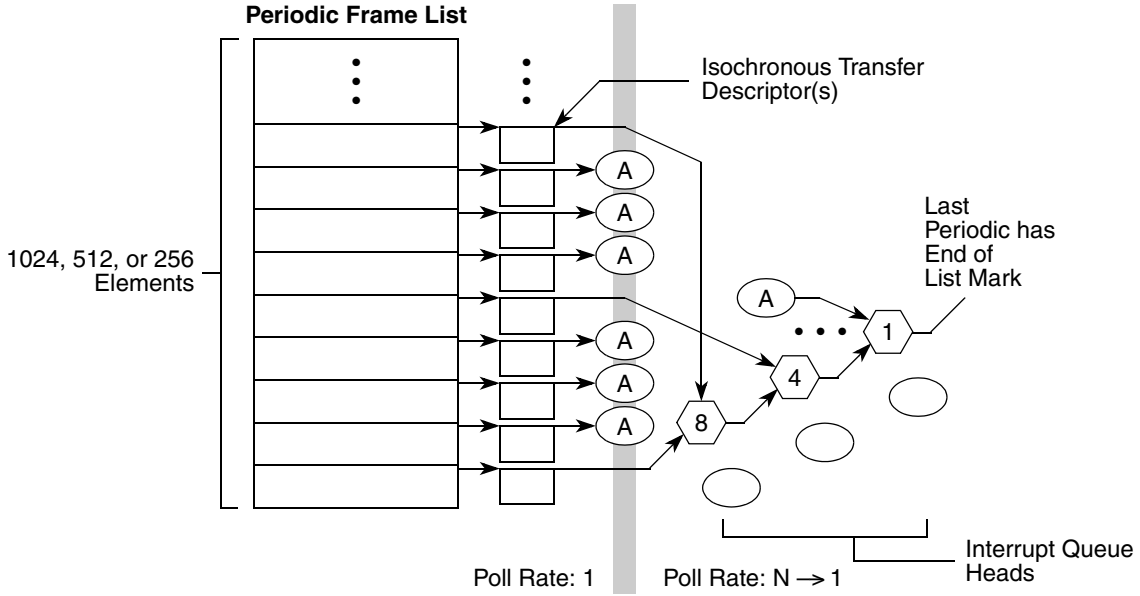


Figure 24-47. Example Periodic Schedule

24.9.8 Managing Isochronous Transfers Using iTDs

The structure of an iTD is presented in Isochronous (High-Speed) Transfer Descriptor (iTD). There are four distinct sections to an iTD:

- The first field is the Next Link Pointer. This field is for schedule linkage purposes only.
- Transaction description array. This area is an eight-element array. Each element represents control and status information for one micro-frame's worth of transactions for a single high-speed isochronous endpoint.
- The buffer page pointer array is a 7-element array of physical memory pointers to data buffers. These are 4K aligned pointers to physical memory.
- Endpoint capabilities. This area utilizes the unused low-order 12 bits of the buffer page pointer array. The fields in this area are used across all transactions executed for this iTD, including endpoint addressing, transfer direction, maximum packet size and high-bandwidth multiplier.

24.9.8.1 Host Controller Operational Model for iTDs

The host controller uses FRINDEX register bits [12:3] to index into the periodic frame list. This means that the host controller visits each frame list element eight consecutive times before incrementing to the next periodic frame list element. Each iTD contains eight transaction descriptions, which map directly to FRINDEX register bits [2:0]. Each iTD can span 8 micro-frames worth of transactions. When the host controller fetches an iTD, it uses FRINDEX register bits [2:0] to index into the transaction description array. If the active bit in the Status field of the indexed transaction description is cleared, the host controller ignores the iTD and follows the Next pointer to the next schedule data structure.

When the indexed active bit is a one the host controller continues to parse the iTD. It stores the indexed transaction description and the general endpoint information (device address, endpoint number, maximum

packet size, etc.). It also uses the Page Select (PG) field to index the buffer pointer array, storing the selected buffer pointer and the next sequential buffer pointer. For example, if PG field is a 0, then the host controller will store Page 0 and Page 1.

The host controller constructs a physical data buffer address by concatenating the current buffer pointer (as selected using the current transaction description's PG field) and the transaction description's Transaction Offset field. The host controller uses the endpoint addressing information and I/O-bit to execute a transaction to the appropriate endpoint. When the transaction is complete, the host controller clears the active bit and writes back any additional status information to the Status field in the currently selected transaction description.

The data buffer associated with the iTD must be virtually contiguous memory. Seven page pointers are provided to support eight high-bandwidth transactions regardless of the starting packet's offset alignment into the first page. A starting buffer pointer (physical memory address) is constructed by concatenating the page pointer (example: page 0 pointer) selected by the active transaction descriptions' PG (example value: 0b00) field with the transaction offset field. As the transaction moves data, the host controller must detect when an increment of the current buffer pointer will cross a page boundary. When this occurs the host controller simply replaces the current buffer pointer's page portion with the next page pointer (example: page 1 pointer) and continues to move data. The size of each bus transaction is determined by the value in the Maximum Packet Size field. An iTD supports high-bandwidth pipes via the Mult (multiplier) field. When the Mult field is 1, 2, or 3, the host controller executes the specified number of Maximum Packet sized bus transactions for the endpoint in the current micro-frame. In other words, the Mult field represents a transaction count for the endpoint in the current micro-frame. If the Mult field is zero, the operation of the host controller is undefined. The transfer description is used to service all transactions indicated by the Mult field.

For OUT transfers, the value of the Transaction *n* Length field represents the total bytes to be sent during the micro-frame. The Mult field must be set by the software to be consistent with Transaction *n* Length and Maximum Packet Size. The host controller will send the bytes in Maximum Packet Size'd portions. After each transaction, the host controller decrements it's local copy of Transaction *n* Length by Maximum Packet Size. The number of bytes the host controller sends is always Maximum Packet Size or Transaction *n* Length, whichever is less. The host controller advances the transfer state in the transfer description, updates the appropriate record in the iTD and moves to the next schedule data structure. The maximum sized transaction supported is 3×1024 bytes.

For IN transfers, the host controller issues Mult transactions. It is assumed that the software has properly initialized the iTD to accommodate all of the possible data. During each IN transaction, the host controller must use Maximum Packet Size to detect packet babble errors. The host controller keeps the sum of bytes received in the Transaction *n* Length field. After all transactions for the endpoint have completed for the micro-frame, Transaction *n* Length contains the total bytes received. If the final value of Transaction *n* Length is less than the value of Maximum Packet Size, then less data than was allowed for was received from the associated endpoint. This short packet condition does not set the USBINT bit in the USBSTS register. The host controller will not detect this condition. If the device sends more than Transaction *n* Length or Maximum Packet Size bytes (whichever is less), then the host controller will set the Babble Detected bit and clear the Active bit. Note, that the host controller is not required to update the iTD field Transaction *n* Length in this error scenario. If the Mult field is greater than one, then the host controller

will automatically execute the value of Mult transactions. The host controller will not execute all Mult transactions if:

- The endpoint is an OUT and Transaction *n* Length goes to zero before all the Mult transactions have executed (ran out of data), or
- The endpoint is an IN and the endpoint delivers a short packet, or an error occurs on a transaction before Mult transactions have been executed. The end of micro-frame may occur before all of the transaction opportunities have been executed. When this happens, the transfer state of the transfer description is advanced to reflect the progress that was made, the result written back to the iTD and the host controller proceeds to processing the next micro-frame.

24.9.8.2 Software Operational Model for iTDs

A client buffer request to an isochronous endpoint may span 1 to N micro-frames. When N is larger than one, the system software may have to use multiple iTDs to read or write data with the buffer (if N is larger than eight, it must use more than one iTD).

Figure 24-48 illustrates the simple model of how a client buffer is mapped by the system software to the periodic schedule (that is, the periodic frame list and a set of iTDs). On the right is the client description of its request. The description includes a buffer base address plus additional annotations to identify which portions of the buffer should be used with each bus transaction. In the middle is the iTD data structures used by the system software to service the client request. Each iTD can be initialized to service up to 24 transactions, organized into eight groups of up to three transactions each. Each group maps to one micro-frame's worth of transactions. The EHCI controller does not provide per-transaction results within a micro-frame. It treats the per-micro-frame transactions as a single logical transfer. On the left is the host controller’s frame list. The system software establishes references from the appropriate locations in the frame list to each of the appropriate iTDs. If the buffer is large, then the system software can use a small set of iTDs to service the entire buffer. The system software can activate the transaction description records (contained in each iTD) in any pattern required for the particular data stream.

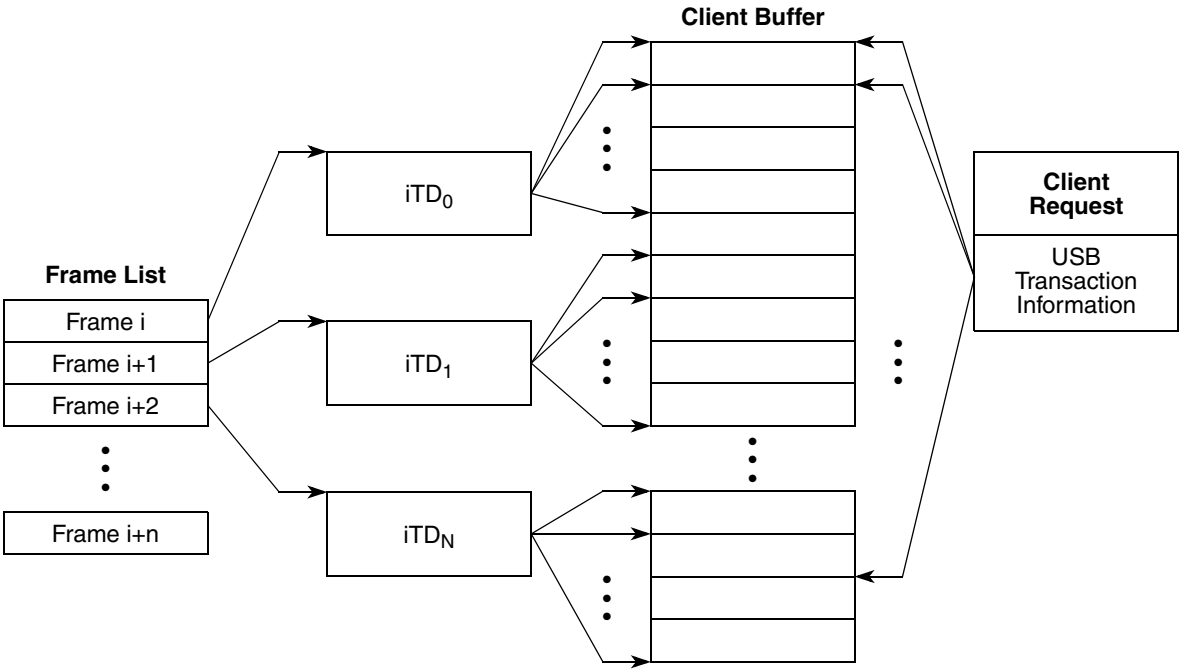


Figure 24-48. Example Association of iTDs to Client Request Buffer

As noted above, the client request includes a pointer to the base of the buffer and offsets into the buffer to annotate which buffer sections are to be used on each bus transaction that occurs on this endpoint. The system software must initialize each transaction description in an iTD to ensure it uses the correct portion of the client buffer. For example, for each transaction description, the PG field is set to index the correct physical buffer page pointer and the Transaction Offset field is set relative to the correct buffer pointer page (for example, the same one referenced by the PG field). When the host controller executes a transaction it selects a transaction description record based on FRINDEX[2:0]. It then uses the current Page Buffer Pointer (as selected by the PG field) and concatenates to the transaction offset field. The result is a starting buffer address for the transaction. As the host controller moves data for the transaction, it must watch for a page wrap condition and properly advance to the next available Page Buffer Pointer. The system software must not use the Page 6 buffer pointer in a transaction description where the length of the transfer will wrap a page boundary. Doing so yields undefined behavior. The host controller hardware is not required to alias the page selector to page zero. USB 2.0 isochronous endpoints can specify a period greater than one. The software can achieve the appropriate scheduling by linking iTDs into the appropriate frames (relative to the frame list) and by setting appropriate transaction description elements active bits to a one.

24.9.8.2.1 Periodic Scheduling Threshold

The Isochronous Scheduling Threshold field in the HCCPARAMS capability register is an indicator to the system software as to how the host controller pre-fetches and effectively caches schedule data structures. It is used by the system software when adding isochronous work items to the periodic schedule. The value of this field indicates to the system software the minimum distance it can update isochronous data (relative to the current location of the host controller execution in the periodic list) and still have the host controller process them.

The iTD and siTD data structures each describe 8 micro-frames worth of transactions. The host controller is allowed to cache one (or more) of these data structures in order to reduce memory traffic. There are three basic caching models that account for the fact the isochronous data structures span 8 micro-frames. The three caching models are: no caching, micro-frame caching and frame caching.

When the software is adding new isochronous transactions to the schedule, it always performs a read of the FRINDEX register to determine the current frame and micro-frame the host controller is currently executing. Of course, there is no information about where in the micro-frame the host controller is, so a constant uncertainty factor of one micro-frame has to be assumed. Combining the knowledge of where the host controller is executing with the knowledge of the caching model allows the definition of simple algorithms for how closely the software can reliably work to the executing host controller.

No caching is indicated with a value of zero in the Isochronous Scheduling Threshold field. The host controller may pre-fetch data structures during a periodic schedule traversal (per micro-frame) but will always dump any accumulated schedule state at the end of the micro-frame. At the appropriate time relative to the beginning of every micro-frame, the host controller always begins schedule traversal from the frame list. The software can use the value of the FRINDEX register (plus the constant 1 uncertainty-factor) to determine the approximate position of the executing host controller. When no caching is selected, the software can add an isochronous transaction as near as 2 micro-frames in front of the current executing position of the host controller.

Frame caching is indicated with a non-zero value in bit [7] of the Isochronous Scheduling Threshold field. In the frame-caching model, the system software assumes that the host controller caches one (or more) isochronous data structures for an entire frame (8 micro-frames). The software uses the value of the FRINDEX register (plus the constant 1 uncertainty) to determine the current micro-frame/frame (assume modulo 8 arithmetic in adding the constant 1 to the micro-frame number). For any current frame N, if the current micro-frame is 0 to 6, then the software can safely add isochronous transactions to Frame N + 1. If the current micro-frame is 7, then software can add isochronous transactions to Frame N + 2.

Micro-frame caching is indicated with a non-zero value in the least-significant 3 bits of the Isochronous Scheduling Threshold field. The system software assumes the host controller caches one or more periodic data structures for the number of micro-frames indicated in the Isochronous Scheduling Threshold field. For example, if the count value were 2, then the host controller keeps a window of 2 micro-frames worth of state (current micro-frame, plus the next) on-chip. On each micro-frame boundary, the host controller releases the current micro-frame state and begins accumulating the next micro-frame state.

24.9.9 Asynchronous Schedule

The Asynchronous schedule traversal is enabled or disabled via the Asynchronous Schedule Enable bit in the USBCMD register. If the Asynchronous Schedule Enable bit is cleared, then the host controller simply does not try to access the asynchronous schedule via the ASYNCLISTADDR register. Likewise, if the Asynchronous Schedule Enable bit is set, the host controller does use the ASYNCLISTADDR register to traverse the asynchronous schedule. Modifications to the Asynchronous Schedule Enable bit are not necessarily immediate. Rather the new value of the bit will only be taken into consideration the next time the host controller needs to use the value of the ASYNCLISTADDR register to get the next queue head.

The Asynchronous Schedule Status bit in the USBSTS register indicates status of the asynchronous schedule. The system software enables (or disables) the asynchronous schedule by writing a one (or zero)

to the Asynchronous Schedule Enable bit in the USBCMD register. The software then can poll the Asynchronous Schedule Status bit to determine when the asynchronous schedule has made the desired transition. The software must not modify the Asynchronous Schedule Enable bit unless the value of the Asynchronous Schedule Enable bit equals that of the Asynchronous Schedule Status bit.

The asynchronous schedule is used to manage all Control and Bulk transfers. Control and Bulk transfers are managed using queue head data structures. The asynchronous schedule is based at the ASYNCLISTADDR register. The default value of the ASYNCLISTADDR register after reset is undefined and the schedule is disabled when the Asynchronous Schedule Enable bit is cleared.

The software may only write this register with defined results when the schedule is disabled, for example, Asynchronous Schedule Enable bit in the USBCMD and the Asynchronous Schedule Status bit in the USBSTS register are cleared. The system software enables execution from the asynchronous schedule by writing a valid memory address (of a queue head) into this register. Then the software enables the asynchronous schedule by setting the Asynchronous Schedule Enable bit is set. The asynchronous schedule is actually enabled when the Asynchronous Schedule Status bit is set.

When the host controller begins servicing the asynchronous schedule, it begins by using the value of the ASYNCLISTADDR register. It reads the first referenced data structure and begins executing transactions and traversing the linked list as appropriate. When the host controller completes processing the asynchronous schedule, it retains the value of the last accessed queue head's horizontal pointer in the ASYNCLISTADDR register. Next time the asynchronous schedule is accessed, this is the first data structure that is serviced. This provides round-robin fairness for processing the asynchronous schedule.

A host controller completes processing the asynchronous schedule when one of the following events occur:

- The end of a micro-frame occurs.
- The host controller detects an empty list condition.
- The schedule has been disabled via the Asynchronous Schedule Enable bit in the USBCMD register.

The queue heads in the asynchronous list are linked into a simple circular list as shown in Figure 24-44. Queue head data structures are the only valid data structures that may be linked into the asynchronous schedule. An isochronous transfer descriptor (iTD or siTD) in the asynchronous schedule yields undefined results.

The maximum packet size field in a queue head is sized to accommodate the use of this data structure for all non-isochronous transfer types. The USB Specification, Revision 2.0 specifies the maximum packet sizes for all transfer types and transfer speeds. The system software should always parameterize the queue head data structures according to the core specification requirements.

24.9.9.1 Adding Queue Heads to Asynchronous Schedule

This is a software requirement section. There are two independent events for adding queue heads to the asynchronous schedule. The first is the initial activation of the asynchronous list. The second is inserting a new queue head into an activated asynchronous list.

Activation of the list is simple. The system software writes the physical memory address of a queue head into the ASYNCLISTADDR register, then enables the list by setting the Asynchronous Schedule Enable bit in the USBCMD register to a one.

When inserting a queue head into an active list, the software must ensure that the schedule is always coherent from the host controllers' point of view. This means that the system software must ensure that all queue head pointer fields are valid. For example qTD pointers have T-Bits set or reference valid qTDs and the Horizontal Pointer references a valid queue head data structure. The following algorithm represents the functional requirements:

```
InsertQueueHead (pQHeadCurrent, pQueueHeadNew)
    --
    -- Requirement: all inputs must be properly initialized.
    --
    -- pQHeadCurrent is a pointer to a queue head that is
    -- already in the active list
    -- pQHeadNew is a pointer to the queue head to be added
    --
    -- This algorithm links a new queue head into a existing
    -- list
    --
    pQueueHeadNew.HorizontalPointer = pQueueHeadCurrent.HorizontalPointer
    pQueueHeadCurrent.HorizontalPointer = physicalAddressOf(pQueueHeadNew)
End InsertQueueHead
```

24.9.9.2 Removing Queue Heads from Asynchronous Schedule

This is a software requirement section. There are two independent events for removing queue heads from the asynchronous schedule. The first is shutting down (deactivating) the asynchronous list. The second is extracting a single queue head from an activated list. The software deactivates the asynchronous schedule by setting the Asynchronous Schedule Enable bit in the USBCMD register to a zero. The software can determine when the list is idle when the Asynchronous Schedule Status bit in the USBSTS register is cleared. The normal mode of operation is that the software removes queue heads from the asynchronous schedule without shutting it down. The software must not remove an active queue head from the schedule. The software should first deactivate all active qTDs, wait for the queue head to go inactive, then remove the queue head from the asynchronous list. The software removes a queue head from the asynchronous list using the following algorithm. The software merely must ensure all of the link pointers reachable by the host controller are kept consistent.

```
UnlinkQueueHead (pQHeadPrevious, pQueueHeadToUnlink, pQHeadNext)
    --
    -- Requirement: all inputs must be properly initialized.
    --
    -- pQHeadPrevious is a pointer to a queue head that
    -- references the queue head to remove
    -- pQHeadToUnlink is a pointer to the queue head to be
    -- removed
    -- pQheadNext is a pointer to a queue head still in the
    -- schedule. Software provides this pointer with the
    -- following strict rules:
    -- if the host software is one queue head, then
    -- pQHeadNext must be the same as
    -- QueueheadToUnlink.HorizontalPointer. If the host
```

```
        -- software is unlinking a consecutive series of
        -- queue heads, QHeadNext must be set by software to
        -- the queue head remaining in the schedule.
        --
        -- This algorithm unlinks a queue head from a circular list
        --
        pQueueHeadPrevious.HorizontalPointer = pQueueHeadToUnlink.HorizontalPointer
        pQueueHeadToUnlink.HorizontalPointer = pQHeadNext
    End UnlinkQueueHead
```

If the software removes the queue head with the H-bit set, it must select another queue head still linked into the schedule and set its H-bit. This should be completed before removing the queue head. The requirement is that the software keep one queue head in the asynchronous schedule, with its H-bit set. At the point the software has removed one or more queue heads from the asynchronous schedule, it is unknown whether the host controller has a cached pointer to them. Similarly, it is unknown how long the host controller might retain the cached information, as it is implementation dependent and may be affected by the actual dynamics of the schedule load. Therefore, once the software has removed a queue head from the asynchronous list, it must retain the coherency of the queue head (link pointers). It cannot disturb the removed queue heads until it knows that the host controller does not have a local copy of a pointer to any of the removed data structures.

The method the software uses to determine when it is safe to modify a removed queue head is to handshake with the host controller. The handshake mechanism allows the software to remove items from the asynchronous schedule, then execute a simple, lightweight handshake that is used by the software as a key that it can free (or reuse) the memory associated the data structures it has removed from the asynchronous schedule.

The handshake is implemented with three bits in the host controller. The first bit is a command bit (Interrupt on Async Advance Doorbell bit in the USBCMD register) that allows the software to inform the host controller that something has been removed from its asynchronous schedule. The second bit is a status bit (Interrupt on Async Advance bit in the USBSTS register) that the host controller sets after it has released all on-chip state that may potentially reference one of the data structures just removed. When the host controller sets this status bit, it also clears the command bit. The third bit is an interrupt enable (Interrupt on Async Advance bit in the USBINTR register) that is matched with the status bit. If the status bit is set and the interrupt enable bit is set, then the host controller asserts a hardware interrupt.

Figure 24-49 illustrates a general example where consecutive queue heads (B and C) are unlinked from the schedule using the algorithm above. Before the unlink operation, the host controller has a copy of queue head A.

The unlink algorithm requires that as the software unlinks each queue head, the unlinked queue head is loaded with the address of a queue head that will remain in the asynchronous schedule.

When the host controller observes that doorbell bit being set, it makes a note of the local reachable schedule information. In this example, the local reachable schedule information includes both queue heads (A & B). It is sufficient that the host controller can set the status bit (and clear the doorbell bit) as soon as it has traversed beyond current reachable schedule information (that is, traversed beyond queue head (B) in this example).

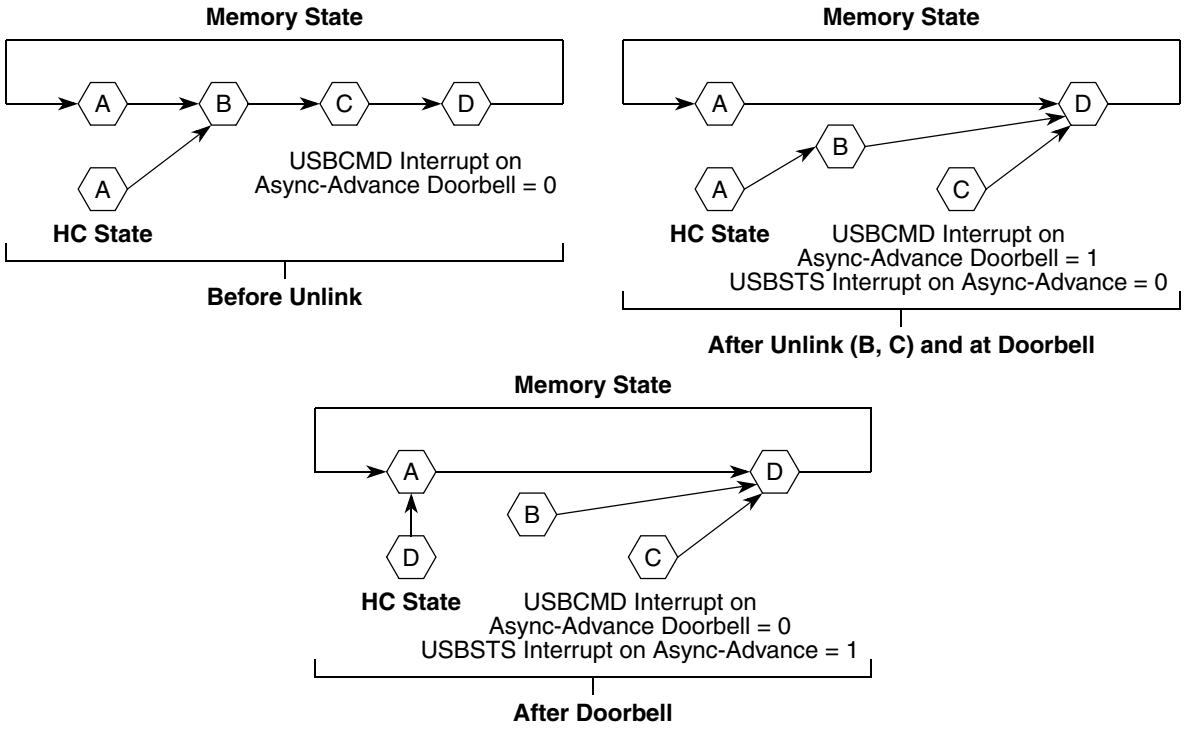


Figure 24-49. Generic Queue Head Unlink Scenario

Alternatively, a host controller implementation is allowed to traverse the entire asynchronous schedule list (for example, observed the head of the queue (twice)) before setting the Advance on Async status bit.

The software may re-use the memory associated with the removed queue heads after it observes the Interrupt on Async Advance status bit is set, following assertion of the doorbell. The software should acknowledge the Interrupt on Async Advance status as indicated in the USBSTS register, before using the doorbell handshake again

24.9.9.3 Empty Asynchronous Schedule Detection

EHCI uses two bits to detect when the asynchronous schedule is empty. The queue head data structure (see Figure 24-41) defines an H-bit in the queue head, which allows the software to mark a queue head as being the head of the reclaim list. host controller also keeps a 1-bit flag in the USBSTS register (Reclamation) that is cleared when the host controller observes a queue head with the H-bit set. The reclamation flag in the status register is set when any USB transaction from the asynchronous schedule is executed (or whenever the asynchronous schedule starts, see Section 24.9.9.4, “Asynchronous Schedule Traversal: Start Event.”

If the controller ever encounters an H-bit of one and a Reclamation bit of zero, the controller simply stops traversal of the asynchronous schedule.

An example illustrating the H-bit in a schedule is shown in Figure 24-50.

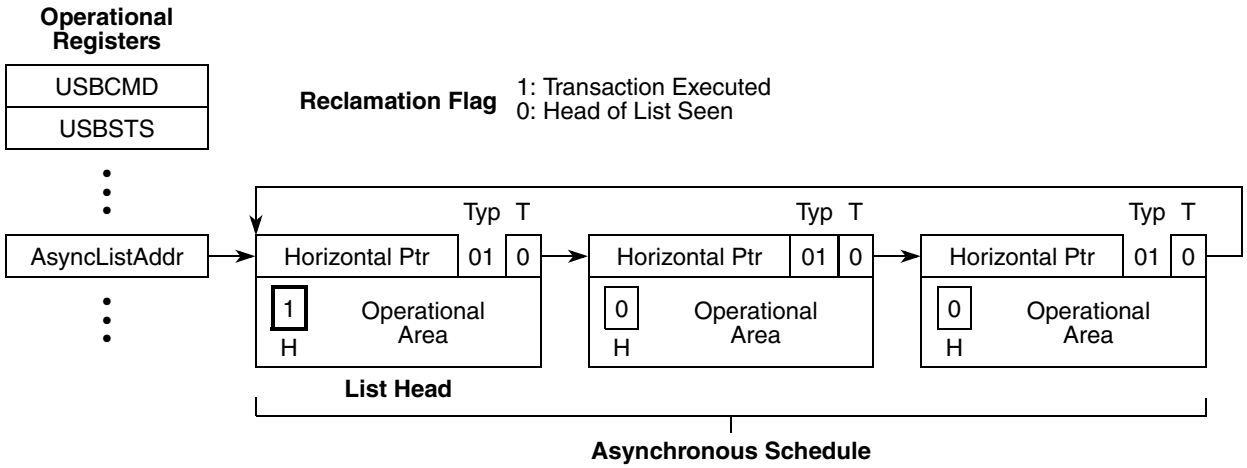


Figure 24-50. Asynchronous Schedule List with Annotation to Mark Head of List

24.9.9.4 Asynchronous Schedule Traversal: Start Event

Once the host controller has idled itself using the empty schedule detection, it naturally activates and begins processing from the Periodic Schedule at the beginning of each micro-frame. In addition, it may have idled itself early in a micro-frame. When this occurs (idles early in the micro-frame) the host controller must occasionally reactivate during the micro-frame and traverse the asynchronous schedule to determine whether any progress can be made. Asynchronous schedule Start Events are defined to be:

- Whenever the host controller transitions from the periodic schedule to the asynchronous schedule. If the periodic schedule is disabled and the asynchronous schedule is enabled, then the beginning of the micro-frame is equivalent to the transition from the periodic schedule, or
- The asynchronous schedule traversal restarts from a sleeping state.

24.9.9.5 Reclamation Status Bit (USBSTS Register)

The operation of the empty asynchronous schedule detection feature depends on the proper management of the Reclamation bit in the USBSTS register. The host controller tests for an empty schedule just after it fetches a new queue head while traversing the asynchronous schedule. The host controller sets the Reclamation bit whenever an asynchronous schedule traversal Start Event occurs. The Reclamation bit is also set whenever the host controller executes a transaction while traversing the asynchronous schedule.The host controller clears the Reclamation bit whenever it finds a queue head with its H-bit set. The software should only set a queue head's H-bit if the queue head is in the asynchronous schedule. If the software sets the H-bit in an interrupt queue head, the resulting behavior is undefined. The host controller may clear the Reclamation bit when executing from the periodic schedule.

24.9.10 Managing Control/Bulk/Interrupt Transfers via Queue Heads

This section presents an overview of how the host controller interacts with queuing data structures.

Queue heads use the Queue Element Transfer Descriptor (qTD) structure defined in Section 24.8.5, “Queue Element Transfer Descriptor (qTD).”

One queue head is used to manage the data stream for one endpoint. The queue head structure contains static endpoint characteristics and capabilities. It also contains a working area from where individual bus transactions for an endpoint are executed. Each qTD represents one or more bus transactions, which is defined in the context of the EHCI specification as a transfer.

The general processing model for the host controller's use of a queue head is simple:

- Read a queue head
- Execute a transaction from the overlay area
- Write back the results of the transaction to the overlay area
- Move to the next queue head

If the host controller encounters errors during a transaction, the host controller will set one of the error reporting bits in the queue head's Status field. The Status field accumulates all errors encountered during the execution of a qTD (that is, the error bits in the queue head Status field are sticky until the transfer (qTD) has completed). This state is always written back to the source qTD when the transfer is complete. On transfer (for example, buffer or halt conditions) boundaries, the host controller must auto-advance (without software intervention) to the next qTD. Additionally, the hardware must be able to halt the queue so no additional bus transactions will occur for the endpoint and the host controller will not advance the queue.

24.9.10.1 Buffer Pointer List Use for Data Streaming with qTDs

A qTD has an array of buffer pointers, which is used to reference the data buffer for a transfer. The EHCI specification requires that the buffer associated with the transfer be virtually contiguous. This means that if the buffer spans more than one physical page, it must obey the following rules:

- The first portion of the buffer must begin at some offset in a page and extend through the end of the page.
- The remaining buffer cannot be allocated in small chunks scattered around memory. For each 4K chunk beyond the first page, each buffer portion matches to a full 4K page. The final portion, which may only be large enough to occupy a portion of a page, must start at the top of the page and be contiguous within that page.

Figure 24-51 illustrates these requirements.

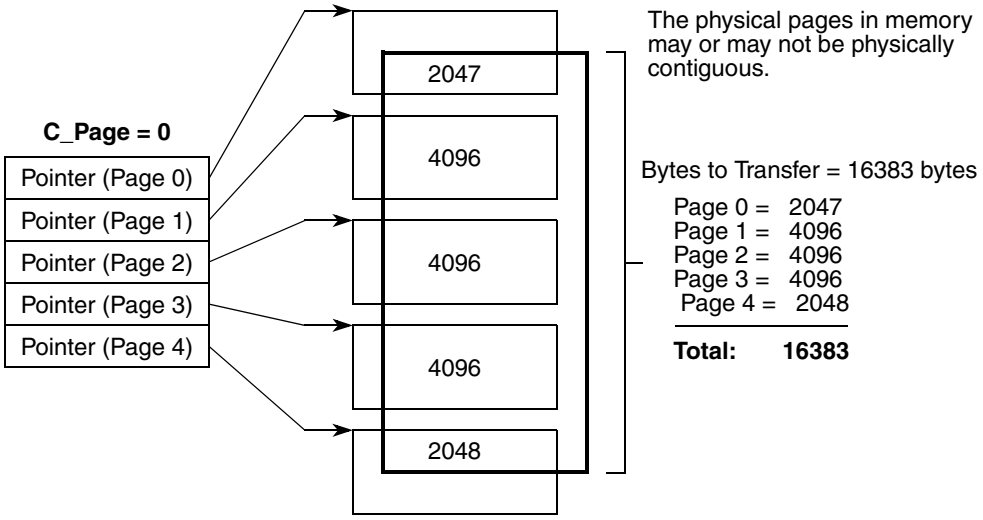


Figure 24-51. Example Mapping of qTD Buffer Pointers to Buffer Pages

The buffer pointer list in the qTD is long enough to support a maximum transfer size of 20K bytes. This case occurs when all five buffer pointers are used and the first offset is zero. A qTD handles a 16Kbyte buffer with any starting buffer alignment.

The host controller uses the C_Page field as an index value to determine which buffer pointer in the list should be used to start the current transaction. The host controller uses a different buffer pointer for each physical page of the buffer. This is always true, even if the buffer is physically contiguous.

The host controller must detect when the current transaction spans a page boundary and automatically move to the next available buffer pointer in the page pointer list. The next available pointer is reached by incrementing C_Page and pulling the next page pointer from the list. The software must ensure there are sufficient buffer pointers to move the amount of data specified in the Bytes to Transfer field.

Figure 24-51 illustrates a nominal example of how System the software would initialize the buffer pointers list and the C_Page field for a transfer size of 16383 bytes. C_Page is cleared. The upper 20-bits of Page 0 references the start of the physical page. Current Offset (the lower 12-bits of queue head Dword 7) holds the offset in the page for example, 2049 (for example, 4096-2047). The remaining page pointers are set to reference the beginning of each subsequent 4K page.

For the first transaction on the qTD (assuming a 512-byte transaction), the host controller uses the first buffer pointer (page 0 because C_Page is cleared) and concatenates the Current Offset field. The 512 bytes are moved during the transaction, the Current Offset and Total Bytes to Transfer are adjusted by 512 and written back to the queue head working area.

During the 4th transaction, the host controller needs 511 bytes in page 0 and one byte in page 1. The host controller will increment C_Page (to 1) and use the page 1 pointer to move the final byte of the transaction. After the 4th transaction, the active page pointer is the page 1 pointer and Current Offset has rolled to one, and both are written back to the overlay area. The transactions continue for the rest of the buffer, with the host controller automatically moving to the next page pointer (that is, C_Page) when necessary. There are three conditions for how the host controller handles C_Page.

- The current transaction does not span a page boundary. The value of C_Page is not adjusted by the host controller.
- The current transaction does span a page boundary. The host controller must detect the page cross condition and advance to the next buffer while streaming data to/from the USB.
- The current transaction completes on a page boundary (that is, the last byte moved for the current transaction is the last byte in the page for the current page pointer). The host controller must increment C_Page before writing back status for the transaction.

Note that the only valid adjustment the host controller may make to C_Page is to increment by one.

24.9.10.2 Adding Interrupt Queue Heads to the Periodic Schedule

The link path(s) from the periodic frame list to a queue head establishes in which frames a transaction can be executed for the queue head. Queue heads are linked into the periodic schedule so they are polled at the appropriate rate. The system software sets a bit in a queue head's S-Mask to indicate which micro-frame within a 1 millisecond period a transaction should be executed for the queue head. The software must ensure that all queue heads in the periodic schedule have S-Mask set to a non-zero value. An S-mask with a zero value in the context of the periodic schedule yields undefined results.

If the desired poll rate is greater than one frame, the system software can use a combination of queue head linking and S-Mask values to spread interrupts of equal poll rates through the schedule so that the periodic bandwidth is allocated and managed in the most efficient manner possible. Some examples are illustrated in Table 24-65.

Table 24-65. Example Periodic Reference Patterns for Interrupt Transfers

Frame # Reference Sequence	Description
0, 2, 4, 6, 8,.... S-Mask = 0x01	A queue head for the bInterval of 2 milliseconds (16 micro-frames) is linked into the periodic schedule so that it is reachable from the periodic frame list locations indicated in the previous column. In addition, the S-Mask field in the queue head is set to 0x01, indicating that the transaction for the endpoint should be executed on the bus during micro-frame 0 of the frame.
0, 2, 4, 6, 8,... S-Mask = 0x02	Another example of a queue head with a bInterval of 2 milliseconds is linked into the periodic frame list at exactly the same interval as the previous example. However, the S-Mask is set to 0x02 indicating that the transaction for the endpoint should be executed on the bus during micro-frame 1 of the frame.

24.9.10.3 Managing Transfer Complete Interrupts from Queue Heads

The host controller sets an interrupt to be signaled at the next interrupt threshold when the completed transfer (qTD) has an Interrupt on Complete (IOC) bit set, or whenever a transfer (qTD) completes with a short packet. If the system software needs multiple qTDs to complete a client request (that is, like a control transfer) the intermediate qTDs do not require interrupts. The system software may only need a single interrupt to notify it that the complete buffer has been transferred. The system software may set IOC's to occur more frequently. A motivation for this may be that it wants early notification so that interface data structures can be re-used in a timely manner.

24.9.11 Ping Control

USB 2.0 defines an addition to the protocol for high-speed devices called Ping. Ping is required for all USB 2.0 High-speed bulk and control endpoints. Ping is not allowed for a split-transaction stream. This extension to the protocol eliminates the bad side-effects of Naking OUT endpoints. The Status field has a Ping State bit, which the host controller uses to determine the next actual PID it will use in the next transaction to the endpoint (see Table 24-53). The Ping State bit is only managed by the host controller for queue heads that meet all of the following criteria:

- The queue head is not an interrupt
- The EPS field equals High-Speed
- The PIDCode field equals OUT

Table 24-66 illustrates the state transition table for the host controller's responsibility for maintaining the PING protocol. Refer to Chapter 8 in the *USB Specification, Revision 2.0* for detailed description on the Ping protocol.

Table 24-66. Ping Control State Transition

	Event		
Current	**Host**	**Device**	**Next**
Do Ping	PING	Nak	Do Ping
Do Ping	PING	Ack	Do OUT
Do Ping	PING	XactErr[1]	Do Ping
Do Ping	PING	Stall	N/C[2]
Do OUT	OUT	Nak	Do Ping
Do OUT	OUT	Nyet	Do Ping[3]
Do OUT	OUT	Ack	Do OUT
Do OUT	OUT	XactErr[1]	Do Ping
Do OUT	OUT	Stall	N/C[2]

1 Transaction Error (XactErr) is any time the host misses the handshake.

2 No transition change required for the Ping State bit. The Stall handshake results in the endpoint being halted (for example, Active cleared and Halt set). Software intervention is required to restart queue.

3 A Nyet response to an OUT means that the device has accepted the data, but cannot receive any more at this time. Host must advance the transfer state and additionally, transition the Ping State bit to Do Ping.

The Ping State bit is described in Table 24-53. The defined ping protocol allows the host to be imprecise on the initialization of the ping protocol (that is, start in Do OUT when we don't know whether there is space on the device or not). The host controller manages the Ping State bit. The system software sets the initial value in the queue head when it initializes a queue head. The host controller preserves the Ping State bit across all queue advancements. This means that when a new qTD is written into the queue head overlay area, the previous value of the Ping State bit is preserved.

24.9.12 Split Transactions

USB 2.0 defines extensions to the bus protocol for managing USB 1.x data streams through USB 2.0 hubs. This section describes how the host controller uses the interface data structures to manage data streams with full- and low-speed devices, connected below a USB 2.0 hub, utilizing the split transaction protocol. Refer to the USB 2.0 Specification for the complete definition of the split transaction protocol. Full- and low-speed devices are enumerated identically as high-speed devices, but the transactions to the full- and low-speed endpoints use the split-transaction protocol on the high-speed bus. The split transaction protocol is an encapsulation of (or wrapper around) the full- or low-speed transaction. The high-speed wrapper portion of the protocol is addressed to the USB 2.0 hub and Transaction Translator below which the full- or low-speed device is attached.

EHCI uses dedicated data structures for managing full-speed isochronous data streams. Control, Bulk and Interrupt are managed using the queuing data structures. The interface data structures need to be programmed with the device address and the Transaction Translator number of the USB 2.0 hub operating as the low-/full-speed host controller for this link. The following sections describe the details of how the host controller processes and manages the split transaction protocol.

24.9.12.1 Split Transactions for Asynchronous Transfers

A queue head in the asynchronous schedule with an EPS field indicating a full-or low-speed device indicates to the host controller that it must use split transactions to stream data for this queue head. All full-speed bulk and full-, low-speed control are managed via queue heads in the asynchronous schedule.

The software must initialize the queue head with the appropriate device address and port number for the transaction translator that is serving as the full-/low-speed host controller for the links connecting the endpoint. The software must also initialize the split transaction state bit (SplitXState) to Do-Start-Split. Finally, if the endpoint is a control endpoint, then system The software must set the Control Transfer Type (C) bit in the queue head to a one. If this is not a control transfer type endpoint, the C bit must be initialized by the software to be a zero. This information is used by the host controller to properly set the Endpoint Type (ET) field in the split transaction bus token. When the C bit is a zero, the split transaction token's ET field is set to indicate a bulk endpoint. When the C bit is a one, the split transaction token's ET field is set to indicate a control endpoint. Refer to Chapter 8 of *USB Specification, Revision 2.0* for details.

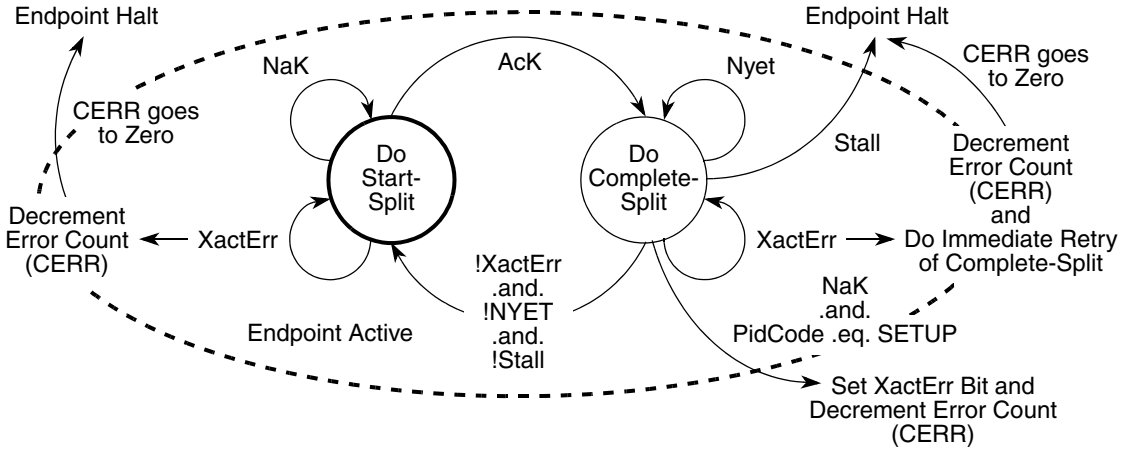


Figure 24-52. Host Controller Asynchronous Schedule Split-Transaction State Machine

24.9.12.1.1 Asynchronous—Do-Start-Split

Do-Start-Split is the state which the software must initialize a full- or low-speed asynchronous queue head. This state is entered from the Do-Complete-Split state only after a complete-split transaction receives a valid response from the transaction translator that is not a Nyet handshake.

For queue heads in this state, the host controller executes a start-split transaction to the transaction translator. If the bus transaction completes without an error and PID Code indicates an IN or OUT transaction, then the host controller reloads the error counter (Cerr). If it is a successful bus transaction and the PID Code indicates a SETUP, the host controller will not reload the error counter. If the transaction translator responds with a Nak, the queue head is left in this state, and the host controller proceeds to the next queue head in the asynchronous schedule.

If the host controller times out the transaction (no response, or bad response) the host controller decrements Cerr and proceeds to the next queue head in the asynchronous schedule.

24.9.12.1.2 Asynchronous—Do-Complete-Split

This state is entered from the Do-Start-Split state only after a start-split transaction receives an Ack handshake from the transaction translator.

For queue heads in this state, the host controller executes a complete-split transaction to the transaction translator. If the transaction translator responds with a Nyet handshake, the queue head is left in this state, the error counter is reset and the host controller proceeds to the next queue head in the asynchronous schedule. When a Nyet handshake is received for a bus transaction where the queue head's PID Code indicates an IN or OUT, the host controller reloads the error counter (Cerr). When a Nyet handshake is received for a complete-split bus transaction where the queue head's PID Code indicates a SETUP, the host controller must not adjust the value of Cerr.

Independent of PID Code, the following responses have the indicated effects:

- Transaction Error (XactErr). Timeout/data CRC failure. The error counter (Cerr) is decremented by one and the complete split transaction is immediately retried (if possible). If there is not enough time in the micro-frame to execute the retry, the host controller ensures that the next time the host controller begins executing from the Asynchronous schedule, it must begin executing from this queue head. If another start-split (for some other endpoint) is sent to the transaction translator before the complete-split is really completed, the transaction translator could dump the results (which were never delivered to the host). This is why the core specification states the retries must be immediate. When the host controller returns to the asynchronous schedule in the next micro-frame, the first transaction from the schedule will be the retry for this endpoint. If Cerr went to zero, the host controller halts the queue.
- NAK. The target endpoint Nak'd the full- or low-speed transaction. The state of the transfer is not advanced and the state is exited. If the PID Code is a SETUP, then the Nak response is a protocol error. The XactErr status bit is set and the Cerr field is decremented.
- STALL. The target endpoint responded with a STALL handshake. The host controller sets the halt bit in the status byte, retires the qTD but does not attempt to advance the queue.

If the PID Code indicates an IN, then any of following responses are expected:

- DATA0/1. On reception of data, the host controller ensures the PID matches the expected data toggle and checks CRC. If the packet is good, the host controller advances the state of the transfer (for example, moves the data pointer by the number of bytes received, decrements the BytesToTransfer field by the number of bytes received, and toggles the dt bit). The host controller then exits this state. The response and advancement of transfer may trigger other processing events, such as retirement of the qTD and advancement of the queue.

  If the data sequence PID does not match the expected, the data is ignored, the transfer state is not advanced and this state is exited.

If the PID Code indicates an OUT/SETUP, then any of following responses are expected:

- ACK. The target endpoint accepted the data, so the host controller must advance the state of the transfer. The Current Offset field is incremented by Maximum Packet Length or Bytes to Transfer, whichever is less. The Bytes To Transfer field is decremented by the same amount and the data toggle bit (dt) is toggled. The host controller then exits this state.

  Advancing the transfer state may cause other processing events such as retirement of the qTD and advancement of the queue.

24.9.12.2 Split Transaction Interrupt

Split-transaction Interrupt-IN/OUT endpoints are managed using the same data structures used for high-speed interrupt endpoints. They both co-exist in the periodic schedule. Queue heads/qTDs offer the set of features required for reliable data delivery, which is characteristic to interrupt transfer types. The split-transaction protocol is managed completely within this defined functional transfer framework. For example, for a high-speed endpoint, the host controller will visit a queue head, execute a high-speed transaction (if criteria are met) and advance the transfer state (or not) depending on the results of the entire transaction. For low- and full-speed endpoints, the details of the execution phase are different (that is, takes more than one bus transaction to complete), but the remainder of the operational framework is intact.

24.9.12.2.1 Split Transaction Scheduling Mechanisms for Interrupt

Full- and low-speed Interrupt queue heads have an EPS field indicating full- or low-speed and have a non-zero S-mask field. The host controller can detect this combination of parameters and assume the endpoint is a periodic endpoint. Low- and full-speed interrupt queue heads require the use of the split transaction protocol. The host controller sets the Endpoint Type (ET) field in the split token to indicate the transaction is an interrupt. These transactions are managed through a transaction translator's periodic pipeline. The software should not set these fields to indicate the queue head is an interrupt unless the queue head is used in the periodic schedule.

The system software manages the per/transaction translator periodic pipeline by budgeting and scheduling exactly during which micro-frames the start-splits and complete-splits for each endpoint will occur. The characteristics of the transaction translator are such that the high-speed transaction protocol must execute during explicit micro-frames, or the data or response information in the pipeline is lost. Figure 24-53 illustrates the general scheduling boundary conditions that are supported by the EHCI periodic schedule and queue head data structure. The S and C_n labels indicate micro-frames where the software can schedule start-splits and complete splits (respectively).

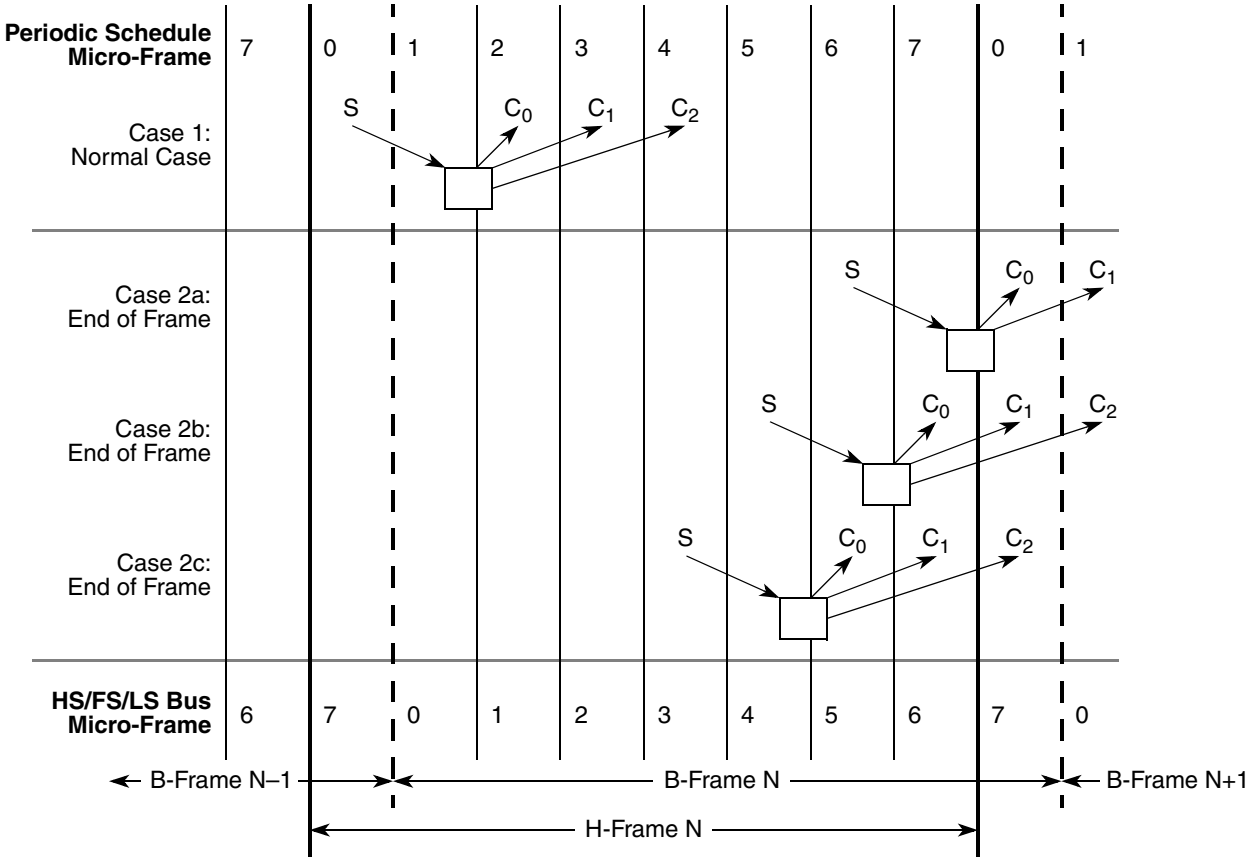


Figure 24-53. Split Transaction, Interrupt Scheduling Boundary Conditions

The scheduling cases are:

- Case 1: The normal scheduling case is where the entire split transaction is completely bounded by a frame (H-Frame in this case).
- Case 2a through Case 2c: The USB 2.0 hub pipeline rules states clearly, when and how many complete-splits must be scheduled to account for earliest to latest execution on the full/low-speed link. The complete-splits may span the H-Frame boundary when the start-split is in micro-frame 4 or later. When this occurs, the H-Frame to B-Frame alignment requires that the queue head be reachable from consecutive periodic frame list locations. The system software cannot build an efficient schedule that satisfies this requirement unless it uses FSTNs. Figure 24-54 illustrates the general layout of the periodic schedule.

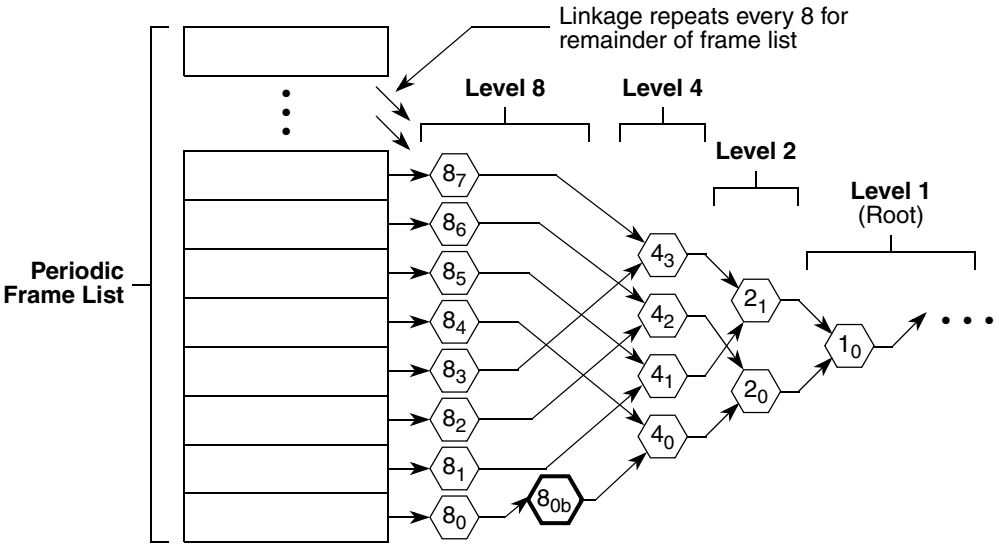


Figure 24-54. General Structure of EHCI Periodic Schedule Utilizing Interrupt Spreading

The periodic frame list is effectively the leaf level a binary tree, which is always traversed leaf to root. Each level in the tree corresponds to a 2^N poll rate. The software can efficiently manage periodic bandwidth on the USB by spreading interrupt queue heads that have the same poll rate requirement across all the available paths from the frame list. For example, system the software can schedule eight poll rate 8 queue heads and account for them once in the high-speed bus bandwidth allocation.

When an endpoint is allocated an execution footprint that spans a frame boundary, the queue head for the endpoint must be reachable from consecutive locations in the frame list. An example would be if 8_{0b} were such an endpoint. Without additional support on the interface, to get 8_{0b} reachable at the correct time, the software would have to link 8_1 to 8_{0b}. It would then have to move 4_1 and everything linked after into the same path as 4_0. This upsets the integrity of the binary tree and disallows the use of the spreading technique.

FSTN data structures are used to preserve the integrity of the binary-tree structure and enable the use of the spreading technique. Section 24.8.7, “Periodic Frame Span Traversal Node (FSTN),” defines the hardware and software operational model requirements for using FSTNs.

The following queue head fields are initialized by the system software to instruct the host controller when to execute portions of the split-transaction protocol.

- SplitXState. This is a single bit residing in the Status field of a queue head (Table 24-53). This bit is used to track the current state of the split transaction.
- Frame S-mask. This is a bit-field where the system software sets a bit corresponding to the micro-frame (within an H-Frame) that the host controller should execute a start-split transaction. This is always qualified by the value of the SplitXState bit in the Status field of the queue head. For example, referring to Figure 24-53, case one, the S-mask would have a value of 0b0000_0001 indicating that if the queue head is traversed by the host controller, and the SplitXState indicates Do_Start, and the current micro-frame as indicated by FRINDEX[2:0] is 0, then execute a start-split transaction.

- Frame C-mask. This is a bit-field where the system software sets one or more bits corresponding to the micro-frames (within an H-Frame) that the host controller should execute complete-split transactions. The interpretation of this field is always qualified by the value of the SplitXState bit in the Status field of the queue head. For example, referring to Figure 24-53, case one, the C-mask would have a value of 0b0001_1100 indicating that if the queue head is traversed by the host controller, and the SplitXState indicates Do_Complete, and the current micro-frame as indicated by FRINDEX[2:0] is 2, 3, or 4, then execute a complete-split transaction. It is the software's responsibility to ensure that the translation between H-Frames and B-Frames is correctly performed when setting bits in S-mask and C-mask.

24.9.12.2.2 Host Controller Operational Model for FSTNs

The FSTN data structure is used to manage Low/Full-speed interrupt queue heads that need to be reached from consecutive frame list locations (that is, boundary cases 2a through 2c). An FSTN is essentially a back pointer, similar in intent to the back pointer field in the siTD data structure.

This feature provides the software a simple primitive to save a schedule position, redirect the host controller to traverse the necessary queue heads in the previous frame, then restore the original schedule position and complete normal traversal.

There are four components to the use of FSTNs:

- FSTN data structure, defined in Section 24.8.7, “Periodic Frame Span Traversal Node (FSTN).”
- A Save Place indicator; this is always an FSTN with its Back Path Link Pointer[T] bit cleared.
- A Restore indicator; this is always an FSTN with its Back Path Link Pointer[T] bit set.
- Host controller FSTN traversal rules.

When the host controller encounters an FSTN during micro-frames 2 through 7 it simply follows the node's Normal Path Link Pointer to access the next schedule data structure. Note that the FSTN's Normal Path Link Pointer[T] bit may set, which the host controller must interpret as the end of periodic list mark.

When the host controller encounters a Save-Place FSTN in micro-frames 0 or 1, it saves the value of the Normal Path Link Pointer and sets an internal flag indicating that it is executing in Recovery Path mode. Recovery Path mode modifies the host controller's rules for how it traverses the schedule and limits which data structures are considered for execution of bus transactions. The host controller continues executing in Recovery Path mode until it encounters a Restore FSTN or it determines that it has reached the end of the micro-frame.

The rules for schedule traversal and limited execution while in Recovery Path mode are:

- Always follow the Normal Path Link Pointer when it encounters an FSTN that is a Save-Place indicator. The host controller must not recursively follow Save-Place FSTNs. Therefore, while executing in Recovery Path mode, it must never follow an FSTN's Back Path Link Pointer.
- Do not process an siTD or iTD data structure; simply follow its Next Link Pointer.
- Do not process a QH (Queue Head) whose EPS field indicates a high-speed device; simply follow its Horizontal Link Pointer.
- When a QH's EPS field indicates a Full/Low-speed device, the host controller only considers it for execution if its SplitXState is DoComplete (note: this applies whether the PID Code indicates an

IN or an OUT). Refer to the *EHCI Specification* for a complete list of additional conditions that must be met in general for the host controller to issue a bus transaction. Note that the host controller must not execute a Start-split transaction while executing in Recovery Path mode. Refer to the *EHCI Specification* for special handling when in Recovery Path mode.

- Stop traversing the recovery path when it encounters an FSTN that is a Restore indicator. The host controller unconditionally uses the saved value of the Save-Place FSTN's Normal Path Link Pointer when returning to the normal path traversal. The host controller must clear the context of executing a Recovery Path when it restores schedule traversal to the Save-Place FSTN's Normal Path Link Pointer.

  If the host controller determines that there is not enough time left in the micro-frame to complete processing of the periodic schedule, it abandons traversal of the recovery path, and clears the context of executing a recovery path. The result is that at the start of the next consecutive micro-frame, the host controller starts traversal at the frame list.

An example traversal of a periodic schedule that includes FSTNs is illustrated in Figure 24-55.

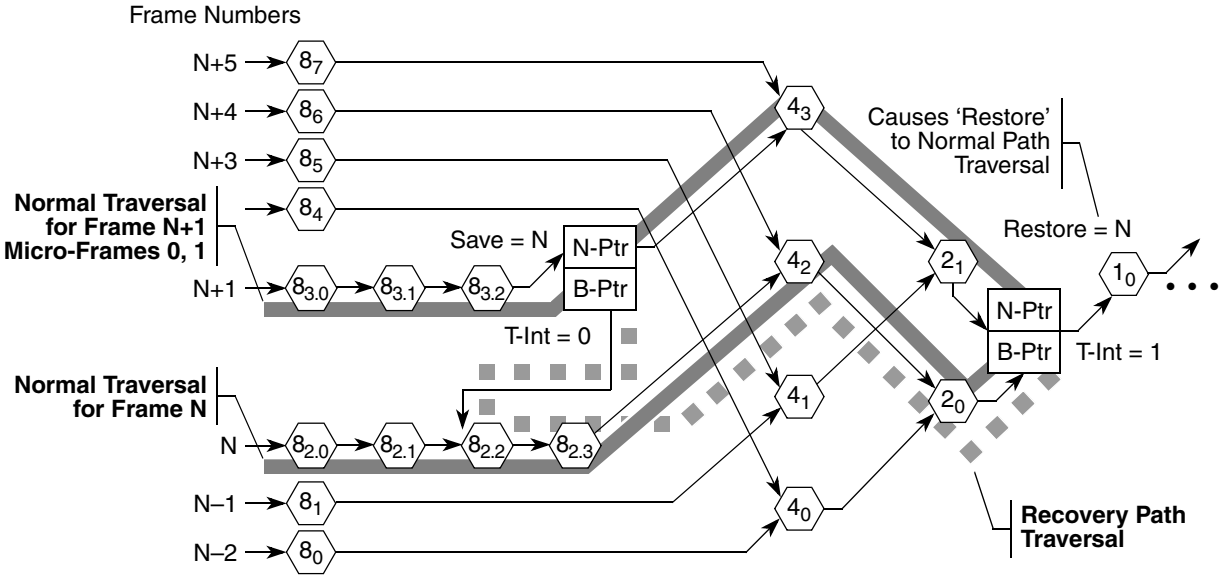


Figure 24-55. Example Host Controller Traversal of Recovery Path via FSTNs

In frame N (micro-frames 0-7), for this example, the host controller traverses all of the schedule data structures utilizing the Normal Path Link Pointers in any FSTNs it encounters. This is because the host controller has not yet encountered a Save-Place FSTN so it is not executing in Recovery Path mode. When it encounters the Restore FSTN, (Restore-N), during micro-frames 0 and 1, it uses Restore-N. Normal Path Link Pointer to traverse to the next data structure (that is, normal schedule traversal). This is because the host controller must use a Restore FSTN's Normal Path Link Pointer when not executing in a Recovery-Path mode. The nodes traversed during frame N include: {$8_{2.0}$, $8_{2.1}$, $8_{2.2}$, $8_{2.3}$, 4_2, 2_0, Restore-N, 1_0...}.

In frame N+1 (micro-frames 0 and 1), when the host controller encounters Save-Path FSTN (Save-N), it observes that Save-N.Back Path Link Pointer.T-bit is zero (definition of a Save-Path indicator). The host controller saves the value of Save-N. Normal Path Link Pointer and follows Save-N.Back Path Link

Pointer. At the same time, it sets an internal flag indicating that it is now in Recovery Path mode (the recovery path is annotated in Figure 24-55 with a large dashed line). The host controller continues traversing data structures on the recovery path and executing only those bus transactions as noted above, on the recovery path until it reaches Restore FSTN (Restore-N). Restore-N.Back Path Link Pointer.T-bit is set (definition of a Restore indicator), so the host controller exits Recovery Path mode by clearing the internal Recovery Path mode flag and commences (restores) schedule traversal using the saved value of the Save-Place FSTN's Normal Path Link Pointer (for example, Save-N.Normal Path Link Pointer). The nodes traversed during these micro-frames include: {$8_{3.0}$, $8_{3.1}$, $8_{3.2}$, Save-A, $8_{2.2}$, $8_{2.3}$, 4_2, 2_0, Restore-N, 4_3, 2_1, Restore-N, 10...}.

In frame N+1 (micro-frames 2-7), when the host controller encounters Save-Path FSTN Save-N, it unconditionally follows Save-N.Normal Path Link Pointer. The nodes traversed during these micro-frames include: {$8_{3.0}$, $8_{3.1}$, $8_{3.2}$, Save-A, 4_3, 2_1, Restore-N, 1_0...}.

24.9.12.2.3 Software Operational Model for FSTNs

The software must create a consistent, coherent schedule for the host controller to traverse. When using FSTNs, the system software must adhere to the following rules:

- Each Save-Place indicator requires a matching Restore indicator.

  The Save-Place indicator is an FSTN with a valid Back Path Link Pointer and T-bit equal to zero. Note that Back Path Link Pointer[Typ] field must be set to indicate the referenced data structure is a queue head. The Restore indicator is an FSTN with its Back Path Link Pointer[T] bit set.

  A Restore FSTN may be matched to one or more Save-Place FSTNs. For example, if the schedule includes a poll-rate 1 level, then the system software only needs to place a Restore FSTN at the beginning of this list in order to match all possible Save-Place FSTNs.
- If the schedule does not have elements linked at a poll-rate level of one, and one or more Save-Place FSTNs are used, then the system software must ensure the Restore FSTN's Normal Path Link Pointer's T-bit is set, as this will be use to mark the end of the periodic list.
- When the schedule does have elements linked at a poll rate level of one, a Restore FSTN must be the first data structure on the poll rate one list. All traversal paths from the frame list converge on the poll-rate one list. The system software must ensure that Recovery Path mode is exited before the host controller is allowed to traverse the poll rate level one list.
- A Save-Place FSTN's Back Path Link Pointer must reference a queue head data structure. The referenced queue head must be reachable from the previous frame list location. In other words, if the Save-Place FSTN is reachable from frame list offset N, then the FSTN's Back Path Link Pointer must reference a queue head that is reachable from frame list offset N-1.

The software should make the schedule as efficient as possible. What this means in this context is that the software should have no more than one Save-Place FSTN reachable in any single frame. Note there will be times when two (or more, depending on the implementation) could exist as full/low-speed footprints change with bandwidth adjustments. This could occur, for example when a bandwidth rebalance causes the system software to move the Save-Place FSTN from one poll rate level to another. During the transition, the software must preserve the integrity of the previous schedule until the new schedule is in place.

24.9.12.2.4 Tracking Split Transaction Progress for Interrupt Transfers

To correctly maintain the data stream, the host controller must be able to detect and report errors where data is lost. For interrupt-IN transfers, data is lost when it makes it into the USB 2.0 hub, but the USB 2.0 host system is unable to get it from the USB 2.0 hub and into the system before it expires from the transaction translator pipeline. When a lost data condition is detected, the queue is halted, thus signaling the system software to recover from the error. A data-loss condition exists whenever a start-split is issued, accepted and successfully executed by the USB 2.0 hub, but the complete-splits get unrecoverable errors on the high-speed link, or the complete-splits do not occur at the correct times. One reason complete-splits might not occur at the right time would be due to host-induced system hold-offs that cause the host controller to miss bus transactions because it cannot get timely access to the schedule in system memory.

The same condition can occur for an interrupt-OUT, but the result is not an endpoint halt condition, but rather effects only the progress of the transfer. The queue head has the following fields to track the progress of each split transaction. These fields are used to keep incremental state about which (and when) portions have been executed.

- C-prog-mask. This is an eight-bit bit-vector where the host controller keeps track of which complete-splits have been executed. Due to the nature of the Transaction Translator periodic pipeline, the complete-splits need to be executed in-order. The host controller needs to detect when the complete-splits have not been executed in order. This can only occur due to system hold-offs where the host controller cannot get to the memory-based schedule. C-prog-mask is a simple bit-vector that the host controller sets one of the C-prog-mask bits for each complete-split executed. The bit position is determined by the micro-frame number in which the complete-split was executed. The host controller always checks C-prog-mask before executing a complete-split transaction. If the previous complete-splits have not been executed then it means one (or more) have been skipped and data has potentially been lost.
- FrameTag. This field is used by the host controller during the complete-split portion of the split transaction to tag the queue head with the frame number (H-Frame number) when the next complete split must be executed.
- S-bytes. This field can be used to store the number of data payload bytes sent during the start-split (if the transaction was an OUT). The S-bytes field must be used to accumulate the data payload bytes received during the complete-splits (for an IN).

24.9.12.2.5 Split Transaction Execution State Machine for Interrupt

In the following section, all references to micro-frame are in the context of a micro-frame within an H-Frame.

As with asynchronous Full- and Low-speed endpoints, a split-transaction state machine is used to manage the split transaction sequence. Aside from the fields defined in the queue head for scheduling and tracking the split transaction, the host controller calculates one internal mechanism that is also used to manage the split transaction. The internal calculated mechanism is:

- cMicroFrameBit. This is a single-bit encoding of the current micro-frame number. It is an eight-bit value calculated by the host controller at the beginning of every micro-frame. It is calculated from the three least significant bits of the FRINDEX register (that is, cMicroFrameBit = (1 shifted-left(FRINDEX[2:0]))). The cMicroFrameBit has at most one bit asserted, which always

corresponds to the current micro-frame number. For example, if the current micro-frame is 0, then cMicroFrameBit will equal 0b0000_0001.

The variable cMicroFrameBit is used to compare against the S-mask and C-mask fields to determine whether the queue head is marked for a start- or complete-split transaction for the current micro-frame.

Figure 24-56 illustrates how a complete interrupt split transaction is managed. There are two phases to each split transaction. The first is a single start-split transaction, which occurs when the SplitXState is at Do_Start and the single bit in cMicroFrameBit has a corresponding bit active in QH[S-mask]. The transaction translator does not acknowledge the receipt of the periodic start-split, so the host controller unconditionally transitions the state to Do_Complete. Due to the available jitter in the transaction translator pipeline, there will be more than one complete-split transaction scheduled by the software for the Do_Complete state. This translates simply to the fact that there are multiple bits set in the QH[C-mask] field.

The host controller keeps the queue head in the Do_Complete state until the split transaction is complete (see definition below), or an error condition triggers the three-strikes-rule (for example, after the host tries the same transaction three times, and each encounters an error, the host controller stops retrying the bus transaction and halts the endpoint, thus requiring the system software to detect the condition and perform system-dependent recovery).

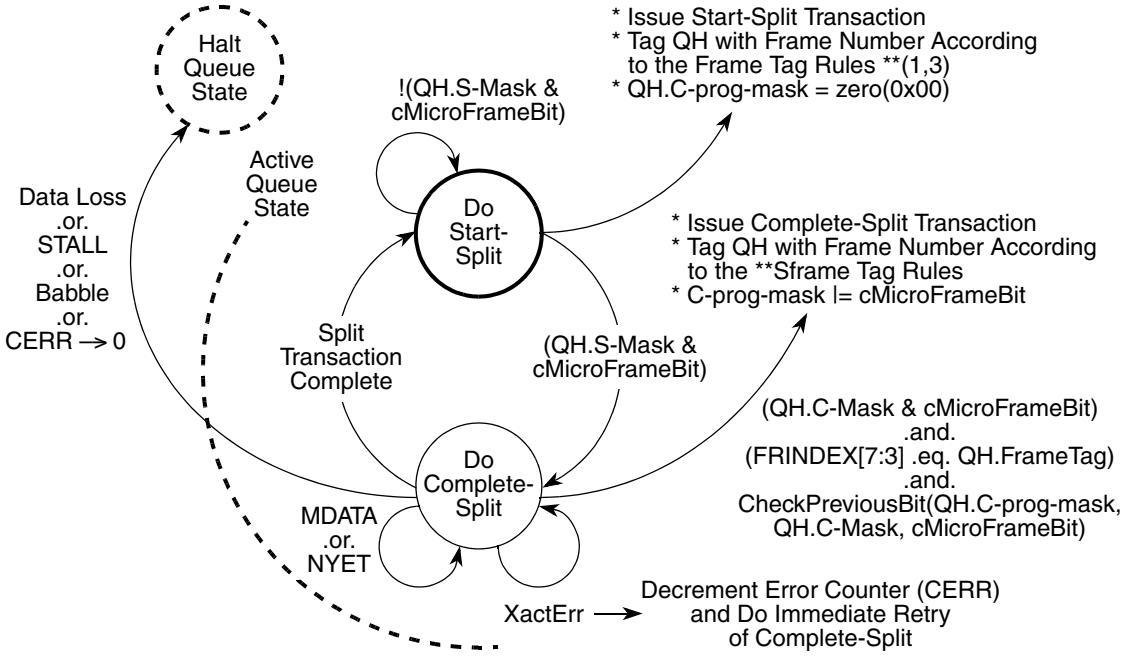


Figure 24-56. Split Transaction State Machine for Interrupt

24.9.12.2.6 Periodic Interrupt—Do-Start-Split

This is the state the software must initialize a full- or low-speed interrupt queue head StartXState bit. This state is entered from the Do_Complete Split state only after the split transaction is complete.

This occurs when one of the following events occur: The transaction translator responds to a complete-split transaction with one of the following:

- NAK. A NAK response is a propagation of the full- or low-speed endpoint's NAK response.
- ACK. An ACK response is a propagation of the full- or low-speed endpoint's ACK response. Only occurs on an OUT endpoint.
- DATA 0/1. Only occurs for INs. Indicates that this is the last of the data from the endpoint for this split transaction.
- ERR. The transaction on the low-/full-speed link below the transaction translator had a failure (for example, timeout, bad CRC, etc.).
- NYET (and Last). The host controller issued the last complete-split and the transaction translator responded with a NYET handshake. This means that the start-split was not correctly received by the transaction translator, so it never executed a transaction to the full- or low-speed endpoint, see Section 24.9.12.2.7, “Periodic Interrupt—Do-Complete-Split” for the definition of 'Last'.

Each time the host controller visits a queue head in this state (once within the Execute Transaction state), bit-wise ANDs QH[S-mask] with cMicroFrameBit to determine whether to execute a start-split. If the result is non-zero, then the host controller issues a start-split transaction. If the PID Code field indicates an IN transaction, the host controller must zero-out the QH[S-bytes] field. After the split-transaction has been executed, the host controller sets up state in the queue head to track the progress of the complete-split phase of the split transaction. Specifically, it records the expected frame number into QH[FrameTag] field, sets C-prog-mask to zero (0x00), and exits this state. Note that the host controller must not adjust the value of Cerr as a result of completion of a start-split transaction.

24.9.12.2.7 Periodic Interrupt—Do-Complete-Split

This state is entered unconditionally from the Do Start Split state after a start-split transaction is executed on the bus. Each time the host controller visits a queue head in this state (once within the Execute Transaction state), it checks to determine whether a complete-split transaction should be executed now.

There are four tests to determine whether a complete-split transaction should be executed.

- Test A. cMicroFrameBit is bit-wise ANDed with QH[C-mask] field. A non-zero result indicates that the software scheduled a complete-split for this endpoint, during this micro-frame.
- Test B. QH[FrameTag] is compared with the current contents of FRINDEX[7:3]. An equal indicates a match.
- Test C. The complete-split progress bit vector is checked to determine whether the previous bit is set, indicating that the previous complete-split was appropriately executed. An example algorithm for this test is provided below:

```
Algorithm Boolean CheckPreviousBit(QH.C-prog-mask, QH.C-mask, cMicroFrameBit)
Begin
-- Return values:
-- TRUE - no error
-- FALSE - error
--
Boolean rvalue = TRUE;
previousBit = cMicroframeBit logical-rotate-right(1)
-- Bit-wise anding previousBit with C-mask indicates
-- whether there was an intent
```

```
-- to send a complete split in the previous micro-frame. So,
-- if the
-- 'previous bit' is set in C-mask, check C-prog-mask to
-- make sure it
-- happened.
If (previousBit bitAND QH.C-mask)then
                                If not(previousBit bitAND QH.C-prog-mask) then
        rvalue = FALSE;
    End if
End If
-- If the C-prog-mask already has a one in this bit position,
-- then an aliasing
-- error has occurred. It will probably get caught by the
-- FrameTag Test, but
-- at any rate it is an error condition that as detectable here
-- should not allow
-- a transaction to be executed.
If (cMicroFrameBit bitAND QH.C-prog-mask) then
rvalue = FALSE;
End if
return (rvalue)
End Algorithm
```

- Test D. Check to see if a start-split should be executed in this micro-frame. Note this is the same test performed in the Do Start Split state. Whenever it evaluates to TRUE and the controller is NOT processing in the context of a Recovery Path mode, it means a start-split should occur in this micro-frame. Test D and Test A are evaluating to TRUE at the same time as a system software error. Behavior is undefined.

If (A.and. B.and. C.and. not(D)) then the host controller will execute a complete-split transaction. When the host controller commits to executing the complete-split transaction, it updates QH[C-prog-mask] by bit-ORing with cMicroFrameBit. On completion of the complete-split transaction, the host controller records the result of the transaction in the queue head and sets QH[FrameTag] to the expected H-Frame number. The effect to the state of the queue head and thus the state of the transfer depends on the response by the transaction translator to the complete-split transaction. The following responses have the effects (note that any responses that result in decrementing of the Cerr will result in the queue head being halted by the host controller if the result of the decrement is zero):

- NYET (and Last). On each NYET response, the host controller checks to determine whether this is the last complete-split for this split transaction. Last is defined in this context as the condition where all of the scheduled complete-splits have been executed. If it is the last complete-split (with a NYET response), then the transfer state of the queue head is not advanced (never received any data) and this state exited. The transaction translator must have responded to all the complete-splits with NYETs, meaning that the start-split issued by the host controller was not received. The start-split should be retried at the next poll period.
- The test for whether this is the Last complete split can be performed by XOR QH[C-mask] with QH[C-prog-mask]. If the result is all zeros then all complete-splits have been executed. When this condition occurs, the XactErr status bit is set and the Cerr field is decremented.
- NYET (and not Last). See above description for testing for Last. The complete-split transaction received a NYET response from the transaction translator. Do not update any transfer state (except for C-prog-mask and FrameTag) and stay in this state. The host controller must not adjust Cerr on this response.

- Transaction Error (XactErr). Timeout, data CRC failure, etc. The Cerr field is decremented and the XactErr bit in the Status field is set. The complete split transaction is immediately retried (if Cerr is non-zero).If there is not enough time in the micro-frame to complete the retry and the endpoint is an IN, or Cerr is decremented to a zero from a one, the queue is halted. If there is not enough time in the micro-frame to complete the retry and the endpoint is an OUT and Cerr is not zero, then this state is exited (that is, return to Do Start Split). This results in a retry of the entire OUT split transaction, at the next poll period. Refer to Chapter 11 Hubs (specifically the section on full- and low-speed interrupts) in the *USB Specification Revision 2.0* for detailed requirements on why these errors must be immediately retried.
- ACK. This can only occur if the target endpoint is an OUT. The target endpoint ACK'd the data and this response is a propagation of the endpoint ACK up to the host controller. The host controller must advance the state of the transfer. The Current Offset field is incremented by Maximum Packet Length or Bytes to Transfer, whichever is less. The field Bytes To Transfer is decremented by the same amount. And the data toggle bit (dt) is toggled. The host controller will then exit this state for this queue head. The host controller must reload Cerr with maximum value on this response. Advancing the transfer state may cause other process events such as retirement of the qTD and advancement of the queue.
- MDATA. This response will only occur for an IN endpoint. The transaction translator responded with zero or more bytes of data and an MDATA PID. The incremental number of bytes received is accumulated in QH[S-bytes]. The host controller must not adjust Cerr on this response.
- DATA0/1. This response may only occur for an IN endpoint. The number of bytes received is added to the accumulated byte count in QH[S-bytes]. The state of the transfer is advanced by the result and the host controller exits this state for this queue head.
- Advancing the transfer state may cause other processing events such as retirement of the qTD and advancement of the queue.
- If the data sequence PID does not match the expected, the entirety of the data received in this split transaction is ignored, the transfer state is not advanced and this state is exited.
- NAK. The target endpoint Nak'd the full- or low-speed transaction. The state of the transfer is not advanced, and this state is exited. The host controller must reload Cerr with maximum value on this response.
- ERR. There was an error during the full- or low-speed transaction. The ERR status bit is set, Cerr is decremented, the state of the transfer is not advanced, and this state is exited.
- STALL. The queue is halted (an exit condition of the Execute Transaction state). The status field bits: Active bit is cleared and the Halted bit is set and the qTD is retired. Responses which are not enumerated in the list or which are received out of sequence are illegal and may result in undefined host controller behavior. The other possible combinations of tests A, B, C, and D may indicate that data or response was lost. Table 24-67 lists the possible combinations and the appropriate action.

Table 24-67. Interrupt IN/OUT Do Complete Split State Execution Criteria

Condition	Action	Description
not(A) not(D)	Ignore QHD	Neither a start nor complete-split is scheduled for the current micro-frame.Host controller should continue walking the schedule.
A not(C)	If PIDCode = IN Halt QHDIf PIDCode = OUT Retry start-split	Progress bit check failed. These means a complete-split has been missed. There is the possibility of lost data. If PID Code is an IN, then the Queue head must be halted.If PID Code is an OUT, then the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not effect Cerr. In either case, set the Missed Micro-frame bit in the status field to a one.
A not(B) C	If PIDCode = IN Halt QHD If PIDCode = OUT Retry start-split	QH.FrameTag test failed. This means that exactly one or more H-Frames have been skipped. This means complete-splits and have missed. There is the possibility of lost data. If PID Code is an IN, then the Queue head must be halted.If PID Code is an OUT, then the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not effect Cerr. In either case, set the Missed Micro-frame bit in the status field to a one.
A B C not(D)	Execute complete-split	This is the non-error case where the host controller executes a complete-split transaction.
D	If PIDCode = IN Halt QHDIf PIDCode = OUT Retry start-split	This is a degenerate case where the start-split was issued, but all of the complete-splits were skipped and all possible intervening opportunities to detect the missed data failed to fire. If PID Code is an IN, then the Queue head must be halted. If PID Code is an OUT, then the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not effect Cerr. In either case, set the Missed Micro-frame bit in the status field to a one. **Note:** When executing in the context of a Recovery Path mode, the host controller is allowed to process the queue head and take the actions indicated above, or it may wait until the queue head is visited in the normal processing mode. Regardless, the host controller must not execute a start-split in the context of a executing in a Recovery Path mode.

24.9.12.2.8 Managing the QH[FrameTag] Field

The QH[FrameTag] field in a queue head is completely managed by the host controller. The rules for setting QH[FrameTag] are simple:

- Rule 1: If transitioning from Do Start Split to Do Complete Split and the current value of FRINDEX[2:0] is 6, QH[FrameTag] is set to FRINDEX[7:3] + 1. This accommodates split transactions whose start-split and complete-splits are in different H-Frames (case 2a, see Figure 24-53).
- Rule 2: If the current value of FRINDEX[2:0] is 7, QH[FrameTag] is set to FRINDEX[7:3] + 1. This accommodates staying in Do Complete Split for cases 2a, 2b, and 2c in Figure 24-53.
- Rule 3: If transitioning from Do_Start Split to Do Complete Split and the current value of FRINDEX[2:0] is not 6, or currently in Do Complete Split and the current value of (FRINDEX[2:0]) is not 7, FrameTag is set to FRINDEX[7:3]. This accommodates all other cases in Figure 24-53.

24.9.12.2.9 Rebalancing the Periodic Schedule

The system software must occasionally adjust a periodic queue head's S-mask and C-mask fields during operation. This need occurs when adjustments to the periodic schedule create a new bandwidth budget and one or more queue head's are assigned new execution footprints (that is, new S-mask and C-mask values).

It is imperative that the system software must not update these masks to new values in the midst of a split transaction. In order to avoid any race conditions with the update, the host controller provides a simple assist to the system software. The system software sets the Inactivate-on-next-Transaction (I) bit to signal the host controller that it intends to update the S-mask and C-mask on this queue head. The system software then waits for the host controller to observe the I-bit is set and transitions the Active bit to a zero. The rules for how and when the host controller clears the Active bit are:

- If the Active bit is cleared, no action is taken. The host controller does not attempt to advance the queue when the I-bit is set.
- If the Active bit is set and the SplitXState is DoStart (regardless of the value of S-mask), the host controller simply clears the Active bit. The host controller is not required to write the transfer state back to the current qTD. Note that if the S-mask indicates that a start-split is scheduled for the current micro-frame, the host controller must not issue the start-split bus transaction; it must clear the Active bit.

The system software must save transfer state before setting the I-bit. This is required so that it can correctly determine what transfer progress (if any) occurred after the I-bit was set and the host controller executed it's final bus-transaction and cleared the Active bit.

After the system software has updated the S-mask and C-mask, it must then reactivate the queue head. Since the Active bit and the I-bit cannot be updated with the same write, the system software needs to use the following algorithm to coherently re-activate a queue head that has been stopped using the I-bit.

1. Set the Halted bit, then
2. Clear the I-bit, then
3. Set the Active bit and clear the Halted bit in the same write.

Setting the Halted bit inhibits the host controller from attempting to advance the queue between the time the I-bit is cleared and the Active bit is set.

24.9.12.3 Split Transaction Isochronous

Full-speed isochronous transfers are managed using the split-transaction protocol through a USB 2.0 transaction translator in a USB 2.0 hub. The host controller utilizes siTD data structure to support the special requirements of isochronous split-transactions. This data structure uses the scheduling model of isochronous TDs (see Section 24.9.8, “Managing Isochronous Transfers Using iTDs,” for the operational model of iTDs) with the contiguous data feature provided by queue heads. This simple arrangement allows a single isochronous scheduling model and adds the additional feature that all data received from the endpoint (per split transaction) must land into a contiguous buffer.

24.9.12.3.1 Split Transaction Scheduling Mechanisms for Isochronous

Full-speed isochronous transactions are managed through a transaction translator's periodic pipeline. As with full- and low-speed interrupt, the system software manages each transaction translator's periodic pipeline by budgeting and scheduling exactly during which micro-frames the start-splits and complete-splits for each full-speed isochronous endpoint occur. The requirements described in Section 24.9.12.2.1, "Split Transaction Scheduling Mechanisms for Interrupt" apply. Figure 24-57 illustrates the general scheduling boundary conditions that are supported by the EHCI periodic schedule. The S_n and C_n labels indicate micro-frames where the software can schedule start- and complete-splits (respectively). The H-Frame boundaries are marked with a large, solid bold vertical line. The B-Frame boundaries are marked with a large, bold, dashed line. The bottom of the figure illustrates the relationship of an siTD to the H-Frame.

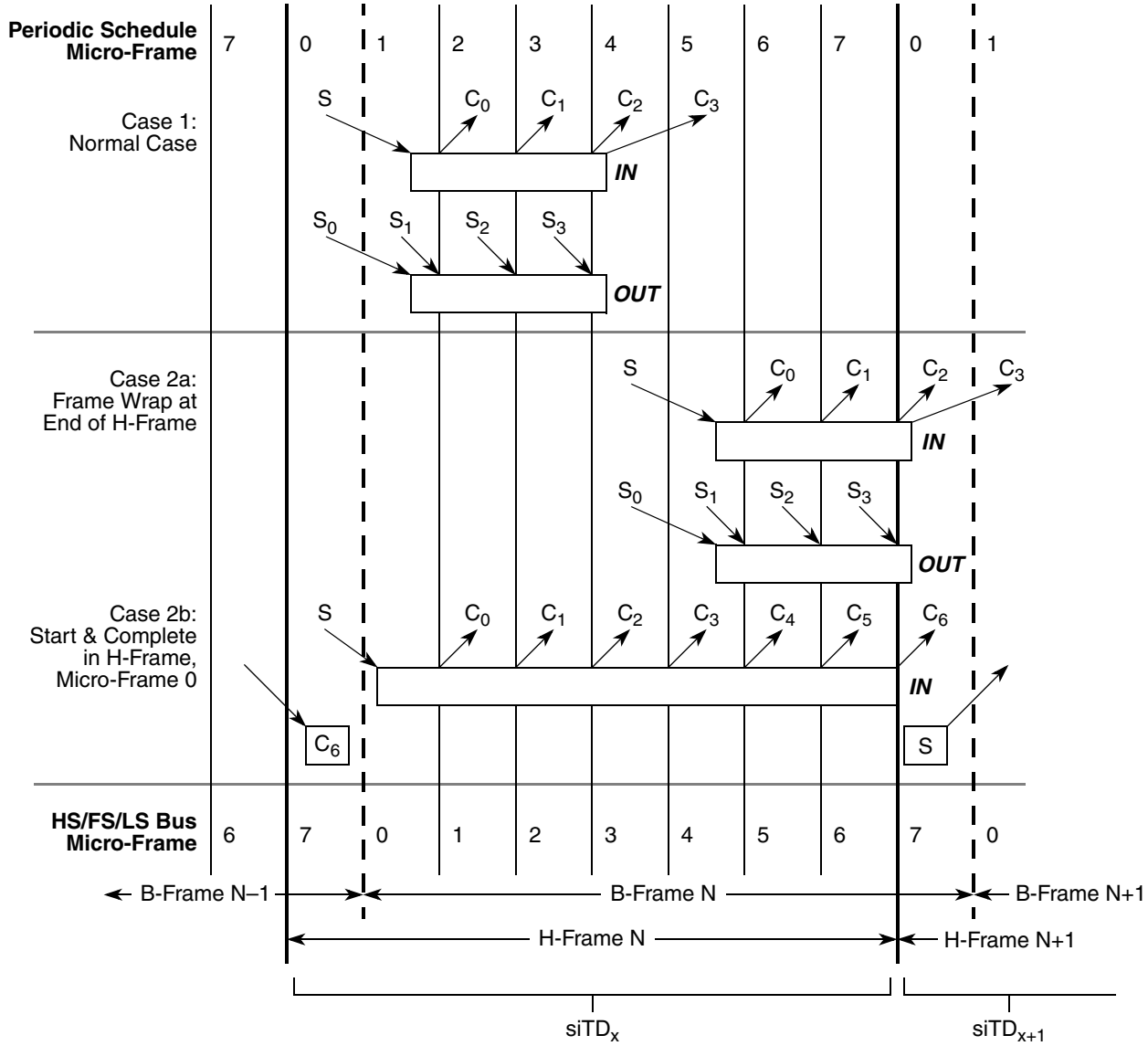


Figure 24-57. Split Transaction, Isochronous Scheduling Boundary Conditions

When the endpoint is an isochronous OUT, there are only start-splits, and no complete-splits. When the endpoint is an isochronous IN, there is at most one start-split and one to N complete-splits. The scheduling boundary cases are:

- Case 1: The entire split transaction is completely bounded by an H-Frame. For example, the start-splits and complete-splits are all scheduled to occur in the same H-Frame.
- Case 2a: This boundary case is where one or more (at most two) complete-splits of a split transaction IN are scheduled across an H-Frame boundary. This can only occur when the split transaction has the possibility of moving data in B-Frame, micro-frames 6 or 7 (H-Frame micro-frame 7 or 0). When an H-Frame boundary wrap condition occurs, the scheduling of the split transaction spans more than one location in the periodic list.(for example, it takes two siTDs in adjacent periodic frame list locations to fully describe the scheduling for the split transaction).

  Although the scheduling of the split transaction may take two data structures, all of the complete-splits for each full-speed IN isochronous transaction must use only one data pointer. For this reason, siTDs contain a back pointer.

  The software must never schedule full-speed isochronous OUTs across an H-Frame boundary.
- Case 2b: This case can only occur for a very large isochronous IN. It is the only allowed scenario where a start-split and complete-split for the same endpoint can occur in the same micro-frame. The software must enforce this rule by scheduling the large transaction first. Large is defined to be anything larger than 579 byte maximum packet size.

A subset of the same mechanisms employed by full- and low-speed interrupt queue heads are employed in siTDs to schedule and track the portions of isochronous split transactions. The following fields are initialized by the system software to instruct the host controller when to execute portions of the split transaction protocol:

- SplitXState. This is a single bit residing in the Status field of an siTD (see Table 24-47). This bit is used to track the current state of the split transaction. The rules for managing this bit are described in Section 24.9.12.3.3, “Split Transaction Execution State Machine for Isochronous.”
- Frame S-mask. This is a bit-field where the system software sets a bit corresponding to the micro-frame (within an H-Frame) that the host controller should execute a start-split transaction. This is always qualified by the value of the SplitXState bit. For example, referring to the IN example in Figure 24-57, case 1, the S-mask would have a value of 0b0000_0001 indicating that if the siTD is traversed by the host controller, and the SplitXState indicates Do Start Split, and the current micro-frame as indicated by FRINDEX[2:0] is 0, then execute a start-split transaction.
- Frame C-mask. This is a bit-field where the system software sets one or more bits corresponding to the micro-frames (within an H-Frame) that the host controller should execute complete-split transactions. The interpretation of this field is always qualified by the value of the SplitXState bit. For example, referring to the IN example in Figure 24-57, case 1, the C-mask would have a value of 0b 0011_1100 indicating that if the siTD is traversed by the host controller, and the SplitXState indicates Do Complete Split, and the current micro-frame as indicated by FRINDEX[2:0] is 2, 3, 4, or 5, then execute a complete-split transaction.
- Back Pointer. This field in a siTD is used to complete an IN split-transaction using the previous H-Frame's siTD. This is only used when the scheduling of the complete-splits span an H-Frame boundary.

There exists a one-to-one relationship between a high-speed isochronous split transaction (including all start- and complete-splits) and one full-speed isochronous transaction. An siTD contains (amongst other things) buffer state and split transaction scheduling information. An siTD's buffer state always maps to one full-speed isochronous data payload. This means that for any full-speed transaction payload, a single siTD's data buffer must be used. This rule applies to both IN an OUTs. An siTD's scheduling information usually also maps to one high-speed isochronous split transaction. The exception to this rule is the H-Frame boundary wrap cases mentioned above.

The siTD data structure describes at most, one frame's worth of high-speed transactions and that description is strictly bounded within a frame boundary. Figure 24-58 illustrates some examples. On the top are examples of the full-speed transaction footprints for the boundary scheduling cases described above. In the middle are time-frame references for both the B-Frames (HS/FS/LS Bus) and the H-Frames. On the bottom is illustrated the relationship between the scope of an siTD description and the time references. Each H-Frame corresponds to a single location in the periodic frame list. The implication is that each siTD is reachable from a single periodic frame list location at a time.

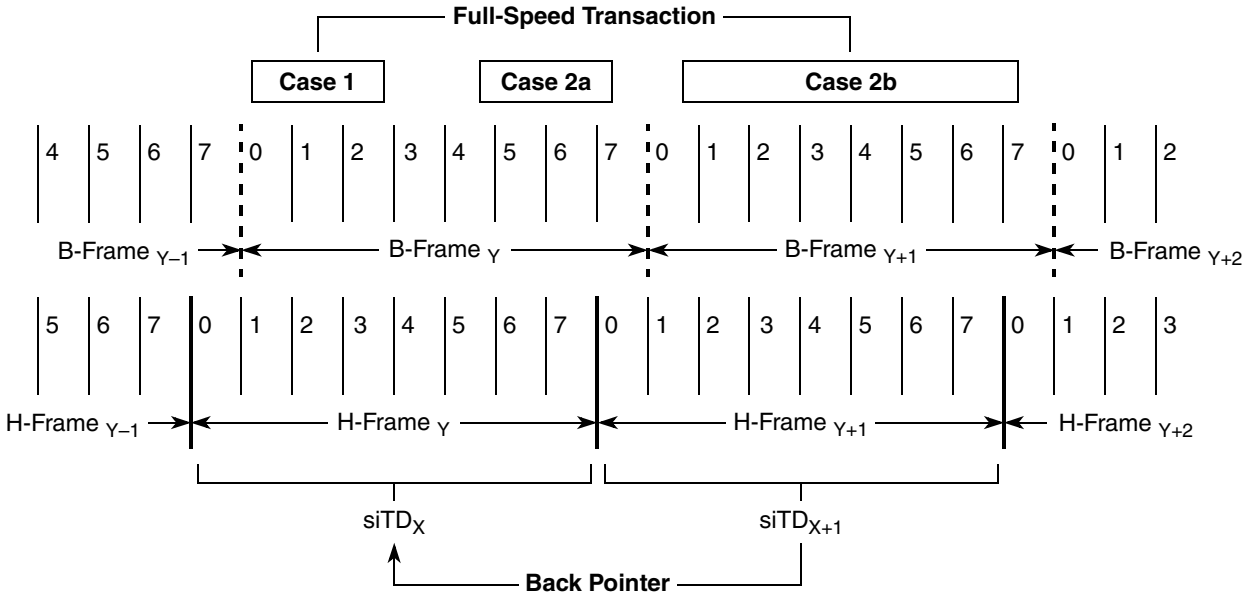


Figure 24-58. siTD Scheduling Boundary Examples

Each case is described as follows:

- Case 1: One siTD is sufficient to describe and complete the isochronous split transaction because the whole isochronous split transaction is tightly contained within a single H-Frame.
- Case 2a, 2b: Although both INs and OUTs can have these footprints, OUTs always take only one siTD to schedule. However, INs (for these boundary cases) require two siTDs to complete the scheduling of the isochronous split transaction. siTDX is used to always issue the start-split and the first N complete-splits. The full-speed transaction (for these cases) can deliver data on the full-speed bus segment during micro-frame 7 of H-Frame$_{Y+1}$, or micro-frame 0 of H-Frame$_{Y+2}$. The complete splits are scheduled using siTD$_{X+2}$ (not shown). The complete-splits to extract this data must use the buffer pointer from siTD$_{X+1}$. The only way for the host controller to reach siTD$_{X+1}$ from H-Frame$_{Y+2}$ is to use siTD$_{X+2}$'s back pointer.

The software must apply the following rules when calculating the schedule and linking the schedule data structures into the periodic schedule:

- The software must ensure that an isochronous split-transaction is started so that it will complete before the end of the B-Frame.
- The software must ensure that for a single full-speed isochronous endpoint, there is never a start-split and complete-split in H-Frame, micro-frame 1. This is mandated as a rule so that case 2a and case 2b can be discriminated. According to the core USB specification, the long isochronous transaction illustrated in Case 2b, could be scheduled so that the start-split was in micro-frame 1 of H-Frame N and the last complete-split would need to occur in micro-frame 1 of H-Frame N+1. However, it is impossible to discriminate between cases 2a and case 2b, which has significant impact on the complexity of the host controller.

24.9.12.3.2 Tracking Split Transaction Progress for Isochronous Transfers

Isochronous endpoints do not employ the concept of a halt on error, however the host controller does identify and report per-packet errors observed in the data stream. This includes schedule traversal problems (skipped micro-frames), timeouts and corrupted data received.

In similar kind to interrupt split-transactions, the portions of the split transaction protocol must execute in the micro-frames they are scheduled. The queue head data structure used to manage full- and low-speed interrupt has several mechanisms for tracking when portions of a transaction have occurred. Isochronous transfers use siTDs for their transfers and the data structures are only reachable using the schedule in the exact micro-frame in which they are required (so all the mechanism employed for tracking in queue heads is not required for siTDs). The software has the option of reusing siTD several times in the complete periodic schedule. However, it must ensure that the results of split transaction N are consumed and the siTD re-initialized (activated) before the host controller gets back to the siTD (in a future micro-frame).

Split-transaction isochronous OUTs utilize a low-level protocol to indicate which portions of the split transaction data have arrived. Control over the low-level protocol is exposed in an siTD using the fields Transaction Position (TP) and Transaction Count (T-count). If the entire data payload for the OUT split transaction is larger than 188 bytes, there will be more than one start-split transaction, each of which require proper annotation. If host hold-offs occur, then the sequence of annotations received from the host will not be complete, which is detected and handled by the transaction translator. See Section 24.9.12.3.1, “Split Transaction Scheduling Mechanisms for Isochronous,” for a description on how these fields are used during a sequence of start-split transactions.

The fields siTD[T-Count] and siTD[TP] are used by the host controller to drive and sequence the transaction position annotations. It is the responsibility of the system software to properly initialize these fields in each siTD. Once the budget for a split-transaction isochronous endpoint is established, S-mask, T-Count, and TP initialization values for all the siTD associated with the endpoint are constant. They remain constant until the budget for the endpoint is recalculated by the software and the periodic schedule adjusted.

For IN-endpoints, the transaction translator simply annotates the response data packets with enough information to allow the host controller to identify the last data. As with split transaction Interrupt, it is the host controller's responsibility to detect when it has missed an opportunity to execute a complete-split. The

following field in the siTD is used to track and detect errors in the execution of a split transaction for an IN isochronous endpoint.

- C-prog-mask. This is an eight-bit bit-vector where the host controller keeps track of which complete-splits have been executed. Due to the nature of the Transaction Translator periodic pipeline, the complete-splits need to be executed in-order. The host controller needs to detect when the complete-splits have not been executed in order. This can only occur due to system hold-offs where the host controller cannot get to the memory-based schedule. C-prog-mask is a simple bit-vector that the host controller sets a bit for each complete-split executed. The bit position is determined by the micro-frame (FRINDEX[2:0]) number in which the complete-split was executed. The host controller always checks C-prog-mask before executing a complete-split transaction. If the previous complete-splits have not been executed, then it means one (or more) have been skipped and data has potentially been lost. The system software is required to initialize this field to zero before setting an siTD's Active bit to a one.

If a transaction translator returns with the final data before all of the complete-splits have been executed, the state of the transfer is advanced so that the remaining complete-splits are not executed. It is important to note that an IN siTD is retired based solely on the responses from the Transaction Translator to the complete-split transactions. This means, for example, that it is possible for a transaction translator to respond to a complete-split with an MDATA PID. The number of bytes in the MDATA's data payload could cause the siTD[Total Bytes to Transfer] field to decrement to zero. This response can occur, before all of the scheduled complete-splits have been executed. In other interface, data structures (for example, high-speed data streams through queue heads), the transition of Total Bytes to Transfer to zero signals the end of the transfer and results in clearing the Active bit. However, in this case, the result has not been delivered by the Transaction Translator and the host must continue with the next complete-split transaction to extract the residual transaction state. This scenario occurs because of the pipeline rules for a Transaction Translator. In summary, the periodic pipeline rules require that on a micro-frame boundary, the Transaction Translator holds the final two bytes received (if it has not seen an End Of Packet (EOP)) in the full-speed bus pipe stage and gives the remaining bytes to the high-speed pipeline stage. At the micro-frame boundary, the Transaction Translator could have received the entire packet (including both CRC bytes) but not received the packet EOP. In the next micro-frame, the Transaction Translator responds with an MDATA and sends all of the data bytes (with the two CRC bytes being held in the full-speed pipeline stage). This could cause the siTD to decrement it's Total Bytes to Transfer field to zero, indicating it has received all expected data. The host must still execute one more (scheduled) complete-split transaction in order to extract the results of the full-speed transaction from the Transaction Translator (for example, the Transaction Translator may have detected a CRC failure, and this result must be forwarded to the host).

If the host experiences hold-offs that cause the host controller to skip one or more (but not all) scheduled split transactions for an isochronous OUT, then the protocol to the transaction translator is not consistent and the transaction translator detects and reacts to the problem. Likewise, for host hold-offs that cause the host controller to skip one or more (but not all) scheduled split transactions for an isochronous IN, the C-prog-mask is used by the host controller to detect errors. However, if the host experiences a hold-off that causes it to skip all of an siTD, or an siTD expires during a host hold off (for example, a hold-off occurs and the siTD is no longer reachable by the host controller in order for it to report the hold-off event), then the system software must detect that the siTDs have not been processed by the host controller (for example, state not advanced) and report the appropriate error to the client driver.

24.9.12.3.3 Split Transaction Execution State Machine for Isochronous

In this section, all references to micro-frame are in the context of a micro-frame within an H-Frame.

If the Active bit in the Status byte is a zero, the host controller ignores the siTD and continues traversing the periodic schedule. Otherwise the host controller processes the siTD as specified below. A split transaction state machine is used to manage the split-transaction protocol sequence. The host controller uses the fields defined in Section 24.9.12.3.2, "Tracking Split Transaction Progress for Isochronous Transfers," plus the variable cMicroFrameBit defined in Section 24.9.12.2.5, "Split Transaction Execution State Machine for Interrupt," to track the progress of an isochronous split transaction. Figure 24-59 illustrates the state machine for managing an siTD through an isochronous split transaction. Bold, dotted circles denote the state of the Active bit in the Status field of a siTD. The Bold, dotted arcs denote the transitions between these states. Solid circles denote the states of the split transaction state machine and the solid arcs denote the transitions between these states. Dotted arcs and boxes reference actions that take place either as a result of a transition or from being in a state.

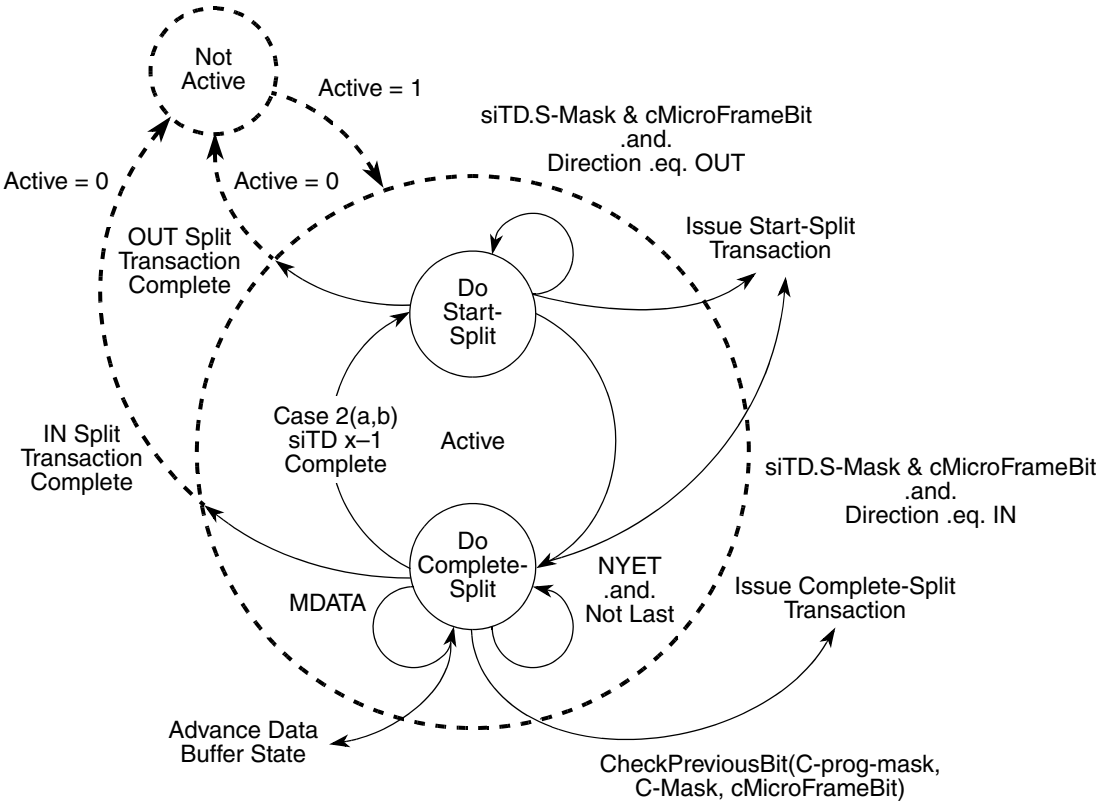


Figure 24-59. Split Transaction State Machine for Isochronous

24.9.12.3.4 Periodic Isochronous—Do-Start-Split

Isochronous split transaction OUTs use only this state. An siTD for a split-transaction isochronous IN is either initialized to this state, or the siTD transitions to this state from Do Complete Split when a case 2a (IN) or 2b scheduling boundary isochronous split-transaction completes.

Each time the host controller reaches an active siTD in this state, it checks the siTD[S-mask] against cMicroFrameBit. If there is a one in the appropriate position, the siTD executes a start-split transaction.

By definition, the host controller cannot reach an siTD at the wrong time. If the I/O field indicates an IN, then the start-split transaction includes only the extended token plus the full-speed token. The software must initialize the siTD[Total Bytes To Transfer] field to the number of bytes expected. This is usually the maximum packet size for the full-speed endpoint. The host controller exits this state when the start-split transaction is complete.

The remainder of this section is specific to an isochronous OUT endpoint (that is, the I/O field indicates an OUT). When the host controller executes a start-split transaction for an isochronous OUT it includes a data payload in the start-split transaction. The memory buffer address for the data payload is constructed by concatenating siTD[Current Offset] with the page pointer indicated by the page select field (siTD[P]). A zero in this field selects Page 0 and a 1 selects Page 1. During the start-split for an OUT, if the data transfer crosses a page boundary during the transaction, the host controller must detect the page cross, update the siTD[P] bit from a zero to a one, and begin using the siTD Page 1 with siTD[Current Offset] as the memory address pointer. The field siTD[TP] is used to annotate each start-split transaction with the indication of which part of the split-transaction data the current payload represents (ALL, BEGIN, MID, END). In all cases, the host controller simply uses the value in siTD[TP] to mark the start-split with the correct transaction position code.

T-Count is always initialized to the number of start-splits for the current frame. TP is always initialized to the first required transaction position identifier. The scheduling boundary case (see Figure 24-58) is used to determine the initial value of TP. The initial cases are summarized in Table 24-68.

Table 24-68. Initial Conditions for OUT siTD's TP and T-Count Fields

Case	T-count	TP	Description
1, 2a	=1	ALL	When the OUT data payload is less than (or equal to) 188 bytes, only one start-split is required to move the data. The one start-split must be marked with an ALL.
1, 2a	!=1	BEGIN	When the OUT data payload is greater than 188 bytes more than one start-split must be used to move the data. The initial start-split must be marked with a BEGIN.

After each start-split transaction is complete, the host controller updates T-Count and TP appropriately so that the next start-split is correctly annotated. Table 24-69 illustrates all of the TP and T-count transitions, which must be accomplished by the host controller.

Table 24-69. Transaction Position (TP)/Transaction Count (T-Count) Transition

TP	T-count Next	TP Next	Description
ALL	0	N/A	Transition from ALL, to done.
BEGIN	1	END	Transition from BEGIN to END. Occurs when T-count starts at 2.
BEGIN	!=1	MID	Transition from BEGIN to MID. Occurs when T-count starts at greater than 2.
MID	!=1	MID	TP stays at MID while T-count is not equal to 1 (for example, greater than 1). This case can occur for any of the scheduling boundary cases where the T-count starts greater than 3.
MID	1	END	Transition from MID to END. This case can occur for any of the scheduling boundary cases where the T-count starts greater than 2.

The start-split transactions do not receive a handshake from the transaction translator, so the host controller always advances the transfer state in the siTD after the bus transaction is complete. To advance the transfer state the following operations take place:

- The siTD[Total Bytes To Transfer] and the siTD[Current Offset] fields are adjusted to reflect the number of bytes transferred.
- The siTD[P] (page select) bit is updated appropriately.
- The siTD[TP] and siTD[T-count] fields are updated appropriately as defined in Table 24-69.

These fields are then written back to the memory based siTD. The S-mask is fixed for the life of the current budget. As mentioned above, TP and T-count are set specifically in each siTD to reflect the data to be sent from this siTD. Therefore, regardless of the value of S-mask, the actual number of start-split transactions depends on T-count (or equivalently, Total Bytes to Transfer). The host controller must clear the Active bit when it detects that all of the schedule data has been sent to the bus. The preferred method is to detect when T-Count decrements to zero as a result of a start-split bus transaction. Equivalently, the host controller can detect when Total Bytes to Transfer decrements to zero. Either implementation must ensure that if the initial condition is Total Bytes to Transfer is equal to zero and T-count is equal to a one, then the host controller will issue a single start-split, with a zero-length data payload. The software must ensure that TP, T-count and Total Bytes to Transfer are set to deliver the appropriate number of bus transactions from each siTD. An inconsistent combination will yield undefined behavior.

If the host experiences hold-offs that cause the host controller to skip start-split transactions for an OUT transfer, the state of the transfer will not progress appropriately. The transaction translator observes protocol violations in the arrival of the start-splits for the OUT endpoint (that is, the transaction position annotation is incorrect as received by the transaction translator).

Example scenarios are described in Section 24.9.12.3.7, “Split Transaction for Isochronous—Processing Examples.”

The host controller can optionally track the progress of an OUT split transaction by setting appropriate bits in the siTD[C-prog-mask] as it executes each scheduled start-split. The checkPreviousBit() algorithm defined in Section 24.9.12.3.5, “Periodic Isochronous—Do Complete Split,” can be used prior to executing each start-split to determine whether start-splits were skipped. The host controller can use this mechanism to detect missed micro-frames. It can then clear the siTD's Active bit and stop execution of this siTD. This saves on both memory and high-speed bus bandwidth.

24.9.12.3.5 Periodic Isochronous—Do Complete Split

This state is only used by a split-transaction isochronous IN endpoint. This state is entered unconditionally from the Do Start State after a start-split transaction is executed for an IN endpoint. Each time the host controller visits an siTD in this state, it conducts a number of tests to determine whether it should execute a complete-split transaction. The individual tests are listed below. The sequence they are applied depends on which micro-frame the host controller is currently executing which means that the tests might not be applied until after the siTD referenced from the back pointer has been fetched.

- Test A. cMicroFrameBit is bit-wise ANDed with the siTD[C-mask] field. A non-zero result indicates that the software scheduled a complete-split for this endpoint, during this micro-frame. This test is always applied to a newly fetched siTD that is in this state.

- Test B. The siTD[C-prog-mask] bit vector is checked to determine whether the previous complete splits have been executed. An example algorithm is given below (this is slightly different than the algorithm used in Section 24.9.12.2.7, "Periodic Interrupt—Do-Complete-Split"). The sequence in which this test is applied depends on the current value of FRINDEX[2:0]. If FRINDEX[2:0] is 0 or 1, it is not applied until the back pointer has been used. Otherwise it is applied immediately.

```
Algorithm Boolean CheckPreviousBit(siTD.C-prog-mask, siTD.C-mask, cMicroFrameBit)
Begin
    Boolean rvalue = TRUE;
    previousBit = cMicroFrameBit rotate-right(1)
    -- Bit-wise anding previousBit with C-mask indicates whether there
    -- was an intent to send a complete split in the previous micro-
    -- frame. So, if the 'previous bit' is set in C-mask, check
    -- C-prog-mask to make sure it happened.
    if previousBit bitAND siTD.C-mask then
        if not (previousBit bitAND siTD.C-prog-mask) then
                                                        rvalue = FALSE
        End if
    End if
    Return rvalue
End Algorithm
```

If Test A is true and FRINDEX[2:0] is zero or one, then this is a case 2a or 2b scheduling boundary (see Figure 24-57). See Section 24.9.12.3.6, "Complete-Split for Scheduling Boundary Cases 2a, 2b," for details in handling this condition.

If Test A and Test B evaluate to true, then the host controller executes a complete-split transaction using the transfer state of the current siTD. When the host controller commits to executing the complete-split transaction, it updates QH[C-prog-mask] by bit-ORing with cMicroFrameBit. The transfer state is advanced based on the completion status of the complete-split transaction. To advance the transfer state of an IN siTD, the host controller must:

- Decrement the number of bytes received from siTD[Total Bytes To Transfer]
- Adjust siTD[Current Offset] by the number of bytes received
- Adjust the siTD[P] (page select) field if the transfer caused the host controller to use the next page pointer
- Set any appropriate bits in the siTD[Status] field, depending on the results of the transaction.

Note that if the host controller encounters a condition where siTD[Total Bytes To Transfer] is zero, and it receives more data, the host controller must not write the additional data to memory. The siTD[Status-Active] bit must be cleared and the siTD[Status-Babble Detected] bit must be set. The fields siTD[Total Bytes To Transfer], siTD[Current Offset], and siTD[P] are not required to be updated as a result of this transaction attempt.

The host controller accepts (assuming good data packet CRC and sufficient room in the buffer as indicated by the value of siTD[Total Bytes To Transfer]) MDATA and DATA0/1 data payloads up to and including 192 bytes. The host controller may optionally clear siTD[Status-Active] and set siTD[Status-Babble Detected] when it receives MDATA or DATA0/1 with a data payload of more than 192 bytes.

The following responses have the noted effects:

- ERR. The full-speed transaction completed with a time-out or bad CRC and this is a reflection of that error to the host. The host controller sets the ERR bit in the siTD[Status] field and clears the Active bit.
- Transaction Error (XactErr). The complete-split transaction encounters a Timeout, CRC16 failure, etc. The siTD[Status] field XactErr field is set and the complete-split transaction must be retried immediately. The host controller must use an internal error counter to count the number of retries as a counter field is not provided in the siTD data structure. The host controller will not retry more than two times. If the host controller exhausts the retries or the end of the micro-frame occurs, the Active bit is cleared.
- DATAx (0 or 1). This response signals that the final data for the split transaction has arrived. The transfer state of the siTD is advanced and the Active bit is cleared. If the Bytes To Transfer field has not decremented to zero (including the reception of the data payload in the DATAx response), then less data than was expected, or allowed for was actually received. This short packet event does not set the USBINT status bit in the USBSTS register to a one. The host controller will not detect this condition.
- NYET (and Last). On each NYET response, the host controller also checks to determine whether this is the last complete-split for this split transaction. Last was defined in Section 24.9.12.2.7, “Periodic Interrupt—Do-Complete-Split.” If it is the last complete-split (with a NYET response), then the transfer state of the siTD is not advanced (never received any data) and the Active bit is cleared. No bits are set in the Status field because this is essentially a skipped transaction. The transaction translator must have responded to all the scheduled complete-splits with NYETs, meaning that the start-split issued by the host controller was not received. This result should be interpreted by the system software as if the transaction was completely skipped. The test for whether this is the last complete split can be performed by XORing C-mask with C-prog-mask. A zero result indicates that all complete-splits have been executed.
- MDATA (and Last). See above description for testing for Last. This can only occur when there is an error condition. Either there has been a babble condition on the full-speed link, which delayed the completion of the full-speed transaction, or the software set up the S-mask and/or C-masks incorrectly. The host controller must set the XactErr bit and clear the Active bit.
- NYET (and not Last). See above description for testing for Last. The complete-split transaction received a NYET response from the transaction translator. Do not update any transfer state (except for C-prog-mask) and stay in this state.
- MDATA (and not Last). The transaction translator responds with an MDATA when it has partial data for the split transaction. For example, the full-speed transaction data payload spans from micro-frame X to X+1 and during micro-frame X, the transaction translator responds with an MDATA and the data accumulated up to the end of micro-frame X. The host controller advances the transfer state to reflect the number of bytes received.

If Test A succeeds, but Test B fails, it means that one or more of the complete-splits have been skipped. The host controller sets the Missed Micro-Frame status bit and clears the Active bit.

24.9.12.3.6 Complete-Split for Scheduling Boundary Cases 2a, 2b

Boundary cases 2a and 2b (INs only) (see Figure 24-57) require that the host controller use the transaction state context of the previous siTD to finish the split transaction. Table 24-70 enumerates the transaction state fields.

Table 24-70. Summary siTD Split Transaction State

Buffer State	Status	Execution Progress
Total Bytes To Transfer P (page select) Current Offset TP (transaction position) T-count (transaction count)	All bits in the status field	C-prog-mask

NOTE

TP and T-count are used only for Host to Device (OUT) endpoints.

If the software has budgeted the schedule of this data stream with a frame wrap case, then it must initialize the siTD[Back Pointer] field to reference a valid siTD and have the T bit in the siTD[Back Pointer] field cleared. Otherwise, the software must set the T bit in siTD[Back Pointer]. The host controller's rules for interpreting when to use the siTD[Back Pointer] field are listed below. These rules apply only when the siTD's Active bit is a one and the SplitXState is Do Complete Split.

- When cMicroFrameBit is a 0x1 and the $siTD_X$[Back Pointer] T-bit is zero, or
- If cMicroFrameBit is a 0x2 and siTDX[S-mask[0]] is zero

When either of these conditions apply, then the host controller must use the transaction state from $siTD_{X-1}$.

In order to access $siTD_{X-1}$, the host controller reads on-chip the siTD referenced from $siTD_X$[Back Pointer].

The host controller must save the entire state from $siTD_X$ while processing $siTD_{X-1}$. This is to accommodate for case 2b processing. The host controller must not recursively walk the list of siTD[Back Pointers].

If $siTD_{X-1}$ is active (Active bit is set and SplitXStat is Do Complete Split), then both Test A and Test B are applied as described above. If these criteria to execute a complete-split are met, the host controller executes the complete split and evaluates the results as described above. The transaction state (see Table 24-70) of $siTD_{X-1}$ is appropriately advanced based on the results and written back to memory. If the resultant state of $siTD_{X-1}$'s Active bit is a one, then the host controller returns to the context of $siTD_X$, and follows its next pointer to the next schedule item. No updates to $siTD_X$ are necessary.

If $siTD_{X-1}$ is active (Active bit is set and SplitXStat is Do Start Split), then the host controller must clear the Active bit and set the Missed Micro-Frame status bit and the resultant status is written back to memory.

If $siTD_{X-1}$'s Active bit is cleared, (because it was cleared when the host controller first visited $siTD_{X-1}$ via $siTD_X$'s back pointer, it transitioned to zero as a result of a detected error, or the results of $siTD_{X-1}$'s complete-split transaction cleared it), then the host controller returns to the context of $siTD_X$ and transitions its SplitXState to Do Start Split. The host controller then determines whether the case 2b start split boundary condition exists (that is, if cMicroframeBit is 1 and $siTD_X$[S-mask[0]] is 1). If this criterion

is met the host controller immediately executes a start-split transaction and appropriately advances the transaction state of $siTD_X$, then follows $siTD_X$[Next Pointer] to the next schedule item. If the criterion is not met, the host controller simply follows $siTD_X$[Next Pointer] to the next schedule item. Note that in the case of a 2b boundary case, the split-transaction of $siTD_{X-1}$ will have its Active bit cleared when the host controller returns to the context of $siTD_X$. Also, note that the software should not initialize an siTD with C-mask bits 0 and 1 set and an S-mask with bit 0 set. This scheduling combination is not supported and the behavior of the host controller is undefined.

24.9.12.3.7 Split Transaction for Isochronous—Processing Examples

There is an important difference between how the hardware/software manages the isochronous split transaction state machine and how it manages the asynchronous and interrupt split transaction state machines. The asynchronous and interrupt split transaction state machines are encapsulated within a single queue head. The progress of the data stream depends on the progress of each split transaction. In some respects, the split-transaction state machine is sequenced using the Execute Transaction queue head traversal state machine.

Isochronous is a pure time-oriented transaction/data stream. The interface data structures are optimized to efficiently describe transactions that need to occur at specific times. The isochronous split-transaction state machine must be managed across these time-oriented data structures. This means that the system software must correctly describe the scheduling of split-transactions across more than one data structure.

Then the host controller must make the appropriate state transitions at the appropriate times, in the correct data structures.

For example, Table 24-71 illustrates a few frames worth of scheduling required to schedule a case 2a full-speed isochronous data stream.

Table 24-71. Example Case 2a—Software Scheduling siTDs for an IN Endpoint

siTDX		Micro-Frames								InitialSplitXState
#	Masks	0	1	2	3	4	5	6	7	
X	S-Mask					1				Do Start Split
	C-Mask	1	1					1	1	
X+1	S-Mask					1				Do Complete Split
	C-Mask	1	1					1	1	
X+2	S-Mask					1				Do Complete Split
	C-Mask	1	1					1	1	
X+3	S-Mask	Repeats previous pattern								Do Complete Split
	C-Mask									

This example shows the first three siTDs for the transaction stream. Since this is the case-2a frame-wrap case, S-masks of all siTDs for this endpoint have a value of 0x10 (a one bit in micro-frame 4) and C-mask value of 0xC3 (one-bits in micro-frames 0,1, 6 and 7). Additionally, the software ensures that the Back

Pointer field of each siTD references the appropriate siTD data structure (and the Back Pointer T-bits are cleared).

The initial SplitXState of the first siTD is Do Start Split. The host controller will visit the first siTD eight times during frame X. The C-mask bits in micro-frames 0 and 1 are ignored because the state is Do Start Split. During micro-frame 4, the host controller determines that it can run a start-split (and does) and changes SplitXState to Do Complete Split. During micro-frames 6 and 7, the host controller executes complete-splits. Notice the siTD for frame X+1 has it's SplitXState initialized to Do Complete Split. As the host controller continues to traverse the schedule during H-Frame X+1, it will visit the second siTD eight times. During micro-frames 0 and 1 it will detect that it must execute complete-splits.

During H-Frame X+1, micro-frame 0, the host controller detects that $siTD_{X+1}$'s Back Pointer[T] bit is a zero, saves the state of $siTD_{X+1}$ and fetches $siTD_X$. It executes the complete split transaction using the transaction state of $siTD_X$. If the $siTD_X$ split transaction is complete, siTD's Active bit is cleared and results written back to $siTD_X$. The host controller retains the fact that $siTD_X$ is retired and transitions the SplitXState in $siTD_{X+1}$ to Do Start Split. At this point, the host controller is prepared to execute the start-split for $siTD_{X+1}$ when it reaches micro-frame 4. If the split-transaction completes early (transaction-complete is defined in Section 24.9.12.3.5, “Periodic Isochronous—Do Complete Split”), that is, before all the scheduled complete-splits have been executed, the host controller changes $siTD_X$[SplitXState] to Do Start Split early and naturally skips the remaining scheduled complete-split transactions. For this example, $siTD_{X+1}$ does not receive a DATA0 response until H-Frame X+2, micro-frame 1.

During H-Frame X+2, micro-frame 0, the host controller detects that $siTD_{X+2}$'s Back Pointer[T] bit is zero, saves the state of $siTD_{X+2}$ and fetches $siTD_{X+1}$. As described above, it executes another split transaction, receives an MDATA response, updates the transfer state, but does not modify the Active bit. The host controller returns to the context of $siTD_{X+2}$, and traverses it's next pointer without any state change updates to $siTD_{X+2}$.

During H-Frame X+2, micro-frame 1, the host controller detects $siTD_{X+2}$'s S-mask[0] bit is zero, saves the state of $siTD_{X+2}$ and fetches $siTD_{X+1}$. It executes another complete-split transaction, receives a DATA0 response, updates the transfer state and clears the Active bit. It returns to the state of $siTD_{X+2}$ and changes its SplitXState to Do Start Split. At this point, the host controller is prepared to execute start-splits for $siTD_{X+2}$ when it reaches micro-frame 4.

24.9.13 Port Test Modes

EHCI host controllers implement the port test modes Test J_State, Test K_State, Test_Packet, Test Force_Enable, and Test SE0_NAK as described in the *USB Specification Revision 2.0*. The required, port test sequence is (assuming the CF-bit in the CONFIGFLAG register is set):

- Disable the periodic and asynchronous schedules by clearing the Asynchronous Schedule Enable and Periodic Schedule Enable bits in the USBCMD register.
- Place all enabled root ports into the suspended state by setting the Suspend bit in each appropriate PORTSC register.
- Clear the Run/Stop bit in the USBCMD register and wait for the HCHalted bit in the USBSTS register, to transition to a one. Note that an EHCI host controller implementation may optionally

allow port testing with the Run/Stop bit set. However, all host controllers must support port testing with Run/Stop cleared and HCHalted set.

- Set the Port Test Control field in the port under test PORTSC register to the value corresponding to the desired test mode. If the selected test is Test_Force_Enable, then the Run/Stop bit in the USBCMD register must then be transitioned back to one, in order to enable transmission of SOFs out of the port under test.
- When the test is complete, the system software must ensure the host controller is halted (HCHalted bit is a one) then it terminates and exits test mode by setting HCReset.

24.9.14 Interrupts

The EHCI host controller hardware provides interrupt capability based on a number of sources. There are several general groups of interrupt sources:

- Interrupts as a result of executing transactions from the schedule (success and error conditions),
- Host controller events (Port change events, etc.), and
- Host controller error events

All transaction-based sources are maskable through the host controller's Interrupt Enable register (USBINTR). Additionally, individual transfer descriptors can be marked to generate an interrupt on completion. This section describes each interrupt source and the processing that occurs in response to the interrupt.

During normal operation, interrupts may be immediate or deferred until the next interrupt threshold occurs. The interrupt threshold is a tunable parameter via the Interrupt Threshold Control field in the USBCMD register. The value of this register controls when the host controller generates an interrupt on behalf of normal transaction execution. When a transaction completes during an interrupt interval period, the interrupt signaling the completion of the transfer will not occur until the interrupt threshold occurs. For example, the default value is eight micro-frames. This means that the host controller will not generate interrupts any more frequently than once every eight micro-frames.

Section 24.9.14.2.4, “Host System Error” details effects of a host system error.

If an interrupt has been scheduled to be generated for the current interrupt threshold interval, the interrupt is not signaled until after the status for the last complete transaction in the interval has been written back to system memory. This may sometimes result in the interrupt not being signaled until the next interrupt threshold.

Initial interrupt processing is the same, regardless of the reason for the interrupt. When an interrupt is signaled by the hardware, CPU control is transferred to host controller's USB interrupt handler. The precise mechanism to accomplish the transfer is OS specific. For this discussion it is just assumed that control is received. When the interrupt handler receives control, its first action is to reads the USBSTS. It then acknowledges the interrupt by clearing all of the interrupt status bits by writing ones to these bit positions. The handler then determines whether the interrupt is due to schedule processing or some other event. After acknowledging the interrupt, the handler (via an OS-specific mechanism), schedules a deferred procedure call (DPC) which will execute later. The DPC routine processes the results of the schedule execution. The precise mechanisms used are beyond the scope of this document.

NOTE

The only method the software should use for acknowledging an interrupt is by transitioning the appropriate status bits in the USBSTS register from a one to a zero.

24.9.14.1 Transfer/Transaction Based Interrupts

These interrupt sources are associated with transfer and transaction progress. They are all dependent on the next interrupt threshold.

24.9.14.1.1 Transaction Error

A transaction error is any error that caused the host controller to think that the transfer did not complete successfully. Table 24-72 lists the events/responses that the host can observe as a result of a transaction. The effects of the error counter and interrupt status are summarized in the following paragraphs. Most of these errors set the XactErr status bit in the appropriate interface data structure.

Table 24-72. Summary of Transaction Errors

Event/ Result	Queue Head/qTD/iTD/siTD Side Effects		USBSTS[USBERRINT]
	Cerr	Status Field	
CRC	-1	XactErr set	1[1]
Timeout	-1	XactErr set	1[1]
Bad PID[2]	-1	XactErr set	1[1]
Babble	N/A	See Section 24.9.14.1.2, “Serial Bus Babble”	1
Buffer Error	N/A	See Section 24.9.14.1.3, “Data Buffer Error”	–

1 If occurs in a queue head, then USBERRINT is asserted only when Cerr counts down from a one to a zero. In addition the queue is halted.

2 The host controller received a response from the device, but it could not recognize the PID as a valid PID.

There is a small set of protocol errors that relate only when executing a queue head and fit under the umbrella of a WRONG PID error that are significant to explicitly identify. When these errors occur, the XactErr status bit in the queue head is set and the Cerr field is decremented. When the PID Code indicates a SETUP, the following responses are protocol errors and result in XactErr bit being set and the Cerr field being decremented.

- EPS field indicates a high-speed device and it returns a Nak handshake to a SETUP.
- EPS field indicates a high-speed device and it returns a Nyet handshake to a SETUP.
- EPS field indicates a low- or full-speed device and the complete-split receives a Nak handshake.

24.9.14.1.2 Serial Bus Babble

When a device transmits more data on the USB than the host controller is expecting for this transaction, it is defined to be babbling. In general, this is called a Packet Babble. When a device sends more data than the Maximum Length number of bytes, the host controller sets the Babble Detected bit to a one and halts

the endpoint if it is using a queue head. Maximum Length is defined as the minimum of Total Bytes to Transfer and Maximum Packet Size. The Cerr field is not decremented for a packet babble condition (only applies to queue heads). A babble condition also exists if IN transaction is in progress at High-speed EOF2 point. This is called a frame babble. A frame babble condition is recorded into the appropriate schedule data structure. In addition, the host controller must disable the port to which the frame babble is detected.

The USBERRINT bit in the USBSTS register is set and if the USB Error Interrupt Enable bit in the USBINTR register is set, then a hardware interrupt is signaled to the system at the next interrupt threshold. The host controller must never start an OUT transaction that babbles across a micro-frame EOF.

NOTE

When a host controller detects a data PID mismatch, it must either: disable the packet babble checking for the duration of the bus transaction or do packet babble checking based solely on Maximum Packet Size. The USB core specification defines the requirements on a data receiver when it receives a data PID mismatch (for example, expects a DATA0 and gets a DATA1 or visa-versa). In summary, it must ignore the received data and respond with an ACK handshake, in order to advance the transmitter's data sequence.The EHCI interface allows the system software to provide buffers for a Control, Bulk or Interrupt IN endpoint that are not an even multiple of the maximum packet size specified by the device. Whenever a device misses an ACK for an IN endpoint, the host and device are out of synchronization with respect to the progress of the data transfer. The host controller may have advanced the transfer to a buffer that is less than maximum packet size. The device re-sends its maximum packet size data packet, with the original data PID, in response to the next IN token. In order to properly manage the bus protocol, the host controller must disable the packet babble check when it observes the data PID mismatch.

24.9.14.1.3 Data Buffer Error

This event indicates that an overrun of incoming data or a underrun of outgoing data has occurred for this transaction. This would generally be caused by the host controller not being able to access required data buffers in memory within necessary latency requirements. These conditions are not considered transaction errors, and do not effect the error count in the queue head. When these errors do occur, the host controller records the fact the error occurred by setting the Data Buffer Error bit in the queue head, iTD or siTD.

If the data buffer error occurs on a non-isochronous IN, the host controller will not issue a handshake to the endpoint. This forces the endpoint to resend the same data (and data toggle) in response to the next IN to the endpoint.

If the data buffer error occurs on an OUT, the host controller must corrupt the end of the packet so that it cannot be interpreted by the device as a good data packet. Simply truncating the packet is not considered acceptable. An acceptable implementation option is to 1's complement the CRC bytes and send them. There are other options suggested in the Transaction Translator section of the *USB Specification Revision 2.0*.

24.9.14.1.4 USB Interrupt (Interrupt on Completion (IOC))

Transfer Descriptors (iTDs, siTDs, and queue heads (qTDs)) contain a bit that can be set to cause an interrupt on their completion. The completion of the transfer associated with that schedule item causes the USB Interrupt (USBINT) bit in the USBSTS register to be set. In addition, if a short packet is encountered on an IN transaction associated with a queue head, then this event also causes USBINT to be set. If the USB Interrupt Enable bit in the USBINTR register is set, a hardware interrupt is signaled to the system at the next interrupt threshold. If the completion is because of errors, the USBERRINT bit in the USBSTS register is also set.

24.9.14.1.5 Short Packet

Reception of a data packet that is less than the endpoint's Max Packet size during Control, Bulk or Interrupt transfers signals the completion of the transfer. Whenever a short packet completion occurs during a queue head execution, the USBINT bit in the USBSTS register is set. If the USB Interrupt Enable bit is set in the USBINTR register, a hardware interrupt is signaled to the system at the next interrupt threshold.

24.9.14.2 Host Controller Event Interrupts

These interrupt sources are independent of the interrupt threshold (with the one exception being the Interrupt on Async Advance.

24.9.14.2.1 Port Change Events

Port registers contain status and status change bits. When the status change bits are set, the host controller sets the Port Change Detect bit in the USBSTS register. If the Port Change Interrupt Enable bit in the USBINTR register is set, then the host controller issues a hardware interrupt. The port status change bits include:

- Connect Status Change
- Port Enable/Disable Change
- Over-current Change
- Force Port Resume

24.9.14.2.2 Frame List Rollover

This event indicates that the host controller has wrapped the frame list. The current programmed size of the frame list effects how often this interrupt occurs. If the frame list size is 1024, then the interrupt occurs every 1024 milliseconds, if it is 512, then it occurs every 512 milliseconds, etc. When a frame list rollover is detected, the host controller sets the Frame List Rollover bit in the USBSTS register. If the Frame List Rollover Enable bit in the USBINTR register is set, the host controller issues a hardware interrupt. This interrupt is not delayed to the next interrupt threshold.

24.9.14.2.3 Interrupt on Async Advance

This event is used for deterministic removal of queue heads from the asynchronous schedule. Whenever the host controller advances the on-chip context of the asynchronous schedule, it evaluates the value of the Interrupt on Async Advance Doorbell bit in the USBCMD register. If it is set, it sets the Interrupt on Async

Advance bit in the USBSTS register. If the Interrupt on Async Advance Enable bit in the USBINTR register is set, the host controller issues a hardware interrupt at the next interrupt threshold. A detailed explanation of this feature is described in Section 24.9.9.2, "Removing Queue Heads from Asynchronous Schedule."

24.9.14.2.4 Host System Error

The host controller is a bus master and any interaction between the host controller and the system may experience errors. The type of host error may be catastrophic to the host controller making it impossible for the host controller to continue in a coherent fashion. Behavior for these types of errors is to halt the host controller. Host-based error must result in the following actions:

- The Run/Stop bit in the USBCMD register is cleared.
- The Host System Error and HCHalted bits in the USBSTS register are set:
- If the Host System Error Enable bit in the USBINTR register is set, then the host controller issues a hardware interrupt. This interrupt is not delayed to the next interrupt threshold.

Table 24-73 summarizes the required actions taken on the various host errors.

Table 24-73. Summary Behavior on Host System Errors

Cycle Type	Master Abort	Target Abort	Data Phase Parity
Frame list pointer fetch (read)	Fatal	Fatal	Fatal
siTD fetch (read)	Fatal	Fatal	Fatal
siTD status write-back (write)	Fatal	Fatal	Fatal
iTD fetch (read)	Fatal	Fatal	Fatal
iTD status write-back (write)	Fatal	Fatal	Fatal
qTD fetch (read)	Fatal	Fatal	Fatal
qHD status write-back (write)	Fatal	Fatal	Fatal
Data write	Fatal	Fatal	Fatal
Data read	Fatal	Fatal	Fatal

NOTE

After a Host System Error, the software must reset the host controller using HCReset in the USBCMD register before re-initializing and restarting the host controller.

24.10 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 data structure definitions in this chapter support a 32-bit memory buffer address space. The interface consists of device Queue Heads and Transfer Descriptors.

NOTE

The software must ensure that no interface data structure reachable by the Device Controller spans a 4K-page boundary.

The data structures defined in the section are (from the device controller's perspective) a mix of read-only and read/ writable fields. The device controller must preserve the read-only fields on all data structure writes.

The USB_DR core includes DCD software called the USB 2.0 Device API. The Device API provides an easy to use Application Program Interface for developing device (peripheral) applications. The Device API incorporates and abstracts for the application developer all of the elements of the program interface.

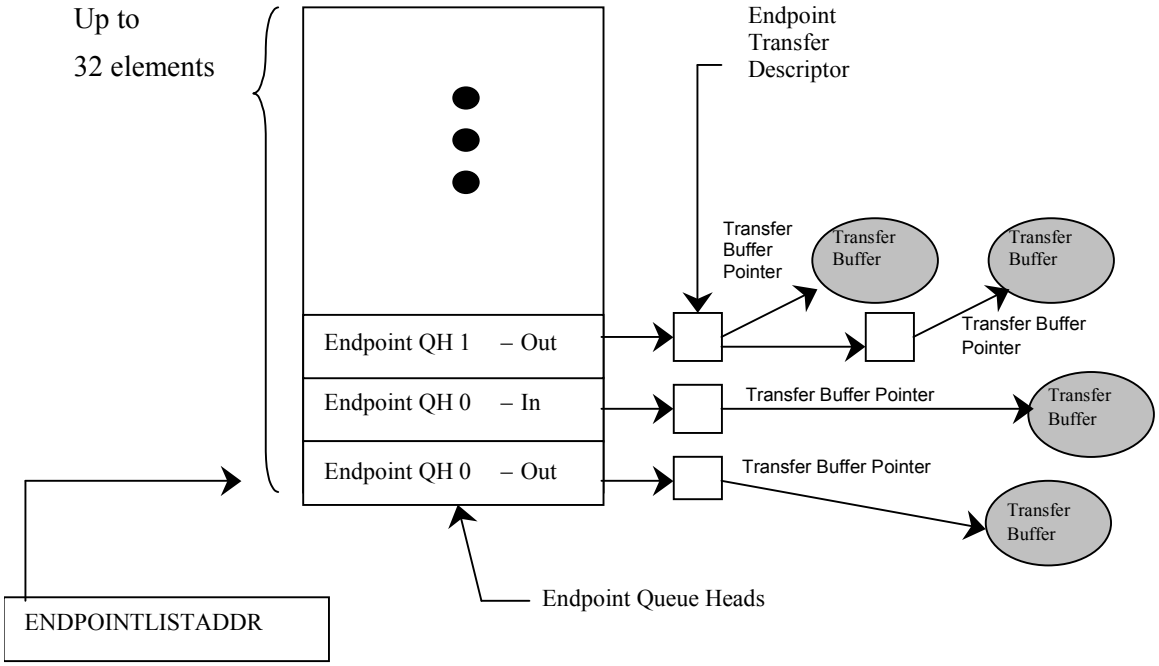


Figure 24-60. End Point Queue Head Organization

Device queue heads are arranged in an array in a continuous area of memory pointed to by the ENDPOINTLISTADDR pointer. The even –numbered device queue heads in the list support receive endpoints (OUT/SETUP) and the odd-numbered queue heads in the list are used for transmit endpoints (IN/INTERRUPT). The device controller will index into this array based upon the endpoint number received from the USB bus. All information necessary to respond to transactions for all primed transfers is contained in this list so the Device Controller can readily respond to incoming requests without having to traverse a linked list.

NOTE

The Endpoint Queue Head List must be aligned to a 2k boundary.

24.10.1 Endpoint Queue Head

The device Endpoint Queue Head (dQH) is where all transfers are managed. The dQH is a 48-byte data structure, but must be aligned on 64-byte boundaries. During priming of an endpoint, the dTD (device

transfer descriptor) is copied into the overlay area of the dQH, which starts at the nextTD pointer DWord and continues through the end of the buffer pointers DWords. After a transfer is complete, the dTD status DWord is updated in the dTD pointed to by the currentTD pointer. While a packet is in progress, the overlay area of the dQH is used as a staging area for the dTD so that the Device Controller can access needed information with little minimal latency.

Figure 24-61 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	00	Maximum Packet Length	ios	000_0000_0000_0000				0x00
Current dTD Pointer[1]								0_0000	0x04
Next dTD Pointer[1]								0000 T[1]	0x08[2]
00	Total Bytes[1]			ioc[1]	000	MultO[1]	00	Status[1]	0x0C[2]
Buffer Pointer (Page 0)[1]						Current Offset[1]			0x10[2]
Buffer Pointer (Page 1)[1]						Reserved			0x14[2]
Buffer Pointer (Page 2)[1]						Reserved			0x18[2]
Buffer Pointer (Page 3)[1]						Reserved			0x1C[2]
Buffer Pointer (Page 4)[1]						Reserved			0x20[2]
Reserved									0x24
Set-up Buffer Bytes 3–0[1]									0x28
Set-up Buffer Bytes 7–4[1]									0x2C

Figure 24-61. Endpoint Queue Head Layout

1 Device controller read/write; all others read-only.

2 Offsets 0x08 through 0x20 contain the transfer overlay.

24.10.1.1 Endpoint Capabilities/Characteristics

This DWord specifies static information about the endpoint, in other words, this information does not change over the lifetime of the endpoint. Device Controller software should not attempt to modify this information while the corresponding endpoint is enabled.

Table 24-74. Endpoint Capabilities/Characteristics

Bit	Name	Description
31–30	Mult	Mult. This field is used to indicate the number of packets executed per transaction description as given by the following: 00 Execute N Transactions as demonstrated by the USB variable length packet protocol where N is computed 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 = 00. Note: ISO endpoints must set Mult = 01, 10, or 11 as needed.

Table 24-74. Endpoint Capabilities/Characteristics (continued)

Bit	Name	Description
29	zlt	Zero length termination select. This bit is used to indicate when a zero length packet is used to terminate transfers where to total transfer length is a multiple. This bit is not relevant for Isochronous transfers. 0 Enable zero length packet to terminate transfers equal to a multiple of the Maximum Packet Length. (default). 1 Disable the zero length packet on transfers that are equal in length to a multiple Maximum Packet Length.
28–27	–	Reserved. These bit reserved for future use and should 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 is used on control type endpoints to indicate if USBINT is set in response to a setup being received.
14–0		Reserved. Bits reserved for future use and should be cleared.

24.10.1.2 Transfer Overlay

The seven DWords 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, then it will not read the associated endpoint.

After an endpoint is readied, the dTD will be copied into this queue head overlay area by the device controller. Until a transfer is expired, the software must not write the queue head overlay area or the associated transfer descriptor. When the transfer is complete, the device controller will write the results back to the original transfer descriptor and advance the queue.

See dTD for a description of the overlay fields.

24.10.1.3 Current dTD Pointer

The current dTD pointer is used by the device controller to locate the transfer in progress. This word is for USB_DR (hardware) use only and should not be modified by DCD software.

Table 24-75. Current dTD Pointer

Bit	Description
31–5	Current dtd. This field is a pointer to the dTD that is represented in the transfer overlay area. This field will be modified by the Device Controller to next dTD pointer during endpoint priming or queue advance.
4–0	Reserved. Bit reserved for future use and should be cleared.

24.10.1.4 Set-Up Buffer

The set-up buffer is dedicated storage for the 8-byte data that follows a set-up PID.

NOTE

Each endpoint has a TX and an RX dQH associated with it, and only the RX queue head is used for receiving setup data packets.

Table 24-76. Multiple Mode Control

DWord	Bits	Description
1	31–0	Setup Buffer 0. This buffer contains bytes 3 to 0 of an incoming setup buffer packet and is written by the device controller to be read by the software.
2	31–0	Setup Buffer 1. This buffer contains bytes 7 to 4 of an incoming setup buffer packet and is written by the device controller to be read by the software.

24.10.2 Endpoint Transfer Descriptor (dTD)

The dTD describes to the device controller the location and quantity of data to be sent/received for given transfer. The DCD should not attempt to modify any field in an active dTD except the Next Link Pointer, which should only be modified as described in Section 24.11.5, "Managing Transfers with Transfer Descriptors."

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12	11 10	9 8	7 6 5 4 3 2 1 0	Offset
Next Link Pointer				0000 T	0x00
00 Total Bytes[1]	ioc 000	MultO	00	Status[1]	0x04
Buffer Pointer (Page 0)		Current Offset[1]			0x08
Buffer Pointer (Page 1)		0	Frame Number[1]		0x0C
Buffer Pointer (Page 2)		0000_0000_0000			0x10
Buffer Pointer (Page 3)		0000_0000_0000			0x14
Buffer Pointer (Page 4)		0000_0000_0000			0x18

Figure 24-62. Endpoint Transfer Descriptor (dTD)

[1] Device controller read/write; all others read-only.

Table 24-77. Next dTD Pointer

Bit	Description
31–5	Next transfer element 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. Bits reserved for future use and should be cleared.
0	Terminate (T). 1=pointer is invalid. 0=Pointer is valid (points to a valid Transfer Element Descriptor). This bit indicates to the Device Controller that there are no more valid entries in the queue.

Table 24-78. dTD Token

Bit	Description
31	Reserved. Bit reserved for future use and should be cleared.
30–16	Total Bytes. This field specifies the total number of bytes to be moved with this transfer descriptor. This field is decremented by the number of bytes actually moved during the transaction and only on the successful completion of the transaction. The maximum value the software may store in the field is 5*4K(5000H). This is the maximum number of bytes 5 page pointers can access. Although it is possible to create a transfer up to 20K this assumes the 1st offset into the first page is 0. When the offset cannot be predetermined, crossing past the 5th page can be guaranteed by limiting the total bytes to 16K**. **Therefore, the maximum recommended transfer is 16K(4000H).** If the value of the field is zero 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. It is not a requirement for IN transfers that Total Bytes To Transfer be an even multiple of Maximum Packet Length. If the software builds such a transfer descriptor for an IN transfer, the last transaction will always be less that Maximum Packet Length.
15	Interrupt On Complete (IOC). This bit is used to indicate if USBINT is to be set in response to device controller being finished with this dTD.
14–12	Reserved. Bits reserved for future use and should be cleared.
11–10	Multiplier Override (MultO). This field can be used for transmit ISO's (that is, ISO-IN) to override the multiplier in the QH. This field must be zero for all packet types that are not transmit-ISO. Example: If QH.multiplier = 3; Maximum packet size = 8; Total Bytes = 15; MultiO = 0 [default] Three packets are sent: {Data2(8); Data1(7); Data0(0)} If QH.multiplier = 3; Maximum packet size = 8; Total Bytes = 15; MultiO = 2 Two packets are sent: {Data1(8); Data0(7)} For maximal efficiency, the software should compute MultO = greatest integer of (Total Bytes / Max. Packet Size) except for the case when Total Bytes = 0; then MultO should be 1. **Note:** Non-ISO and Non-TX endpoints must set MultO= 00.
9–8	Reserved. Bits reserved for future use and should be cleared.
7–0	Status. This field is used by the Device Controller to communicate individual command execution states back to the Device Controller software. This field contains the status of the last transaction performed on this qTD. The bit encodings are: Bit Status Field Description 7 Active. 6 Halted. 5 Data Buffer Error. 3 Transaction Error. 4,2,0 Reserved.

Table 24-79. Buffer Page Pointer List

Bit	Description
31–12	Buffer Pointer. Selects the page offset in memory for the packet buffer. Non virtual memory systems will typically set the buffer pointers to a series of incrementing integers.
0;11–0	Current Offset. Offset into the 4kb buffer where the packet is to begin.
1;10–0	Frame Number. Written by the device controller to indicate the frame number in which a packet finishes. This is typically be used to correlate relative completion times of packets on an ISO endpoint.

24.11 Device Operational Model

The function of the device operation is to transfer a request in the memory image to and from the Universal Serial Bus. Using a set of linked list transfer descriptors, pointed to by a queue head, the device controller will perform the data transfers. The following 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.

24.11.1 Device Controller Initialization

After a hardware reset, the USB OTG controller is disabled until the Run/Stop bit is set to a '1'. In the disabled state, the pull-up on the USB D+ is not active which prevents an attach event from occurring. At a minimum, it is necessary to have the queue heads setup for endpoint zero before the device attach occurs. Shortly after the device is enabled, a USB reset will occur followed by setup packet arriving at endpoint 0. A Queue head must be prepared so that the device controller can store the incoming setup packet.

In order to initialize a device, the software should perform the following steps:

1. Set Controller Mode to device mode. Optionally set Streaming Disable in the USBMODE register.

   NOTE

   Transitioning from host mode to device mode requires a device controller reset before modifying USBMODE.

2. Optionally modify the BURSTSIZE register.
3. Allocate and Initialize device queue heads in system memory Minimum: Initialize device queue heads 0 Tx and 0 Rx.

   NOTE

   All device queue heads must be initialized for control endpoints before the endpoint is enabled. Device queue heads for non-control endpoints must be initialized before the endpoint can be used.

   For information on device queue heads, refer to Section 24.10, “Device Data Structures.”
4. Configure ENDPOINTLISTADDR Pointer.

   For additional information on ENDPOINTLISTADDR, refer Table 24-23.
5. Enable the microprocessor interrupt associated with the USB OTG and optionally change setting of ITC field in USBCMD register.

Recommended: enable all device interrupts including: USBINT, USBERRINT, Port Change Detect, USB Reset Received, DCSuspend.

For a list of available interrupts refer to USBINTR register description Table 24-17 and the USBSTS register description Table 24-16.

6. Set Run/Stop bit to Run Mode.

   After the Run bit is set, a device reset will occur. The DCD must monitor the reset event and set the DEVICEADDR register, set the ENDPTCTRLx registers, and adjust the software state as described in Section 24.11.2.1, “Bus Reset.

NOTE

Endpoint 0 is designed as a control endpoint only and does not need to be configured using ENDPTCTRL0 register.

It is also not necessary to initially prime Endpoint 0 because the first packet received will always be a setup packet. The contents of the first setup packet will require a response in accordance with the USB 2.0 Specification, Chapter 9, *Device Framework,* command set.

24.11.2 Port State and Control

From a chip or system reset, the USB_DR enters the *powered* state. A transition from the powered state to the *attach* state occurs when the Run/Stop bit is set to a '1'. After receiving a reset on the bus, the port will enter the *defaultFS or defaultHS* state in accordance with the protocol reset described in Appendix C.2 of the USB Specification Rev. 2.0. The following state diagram depicts the state of a USB 2.0 device.

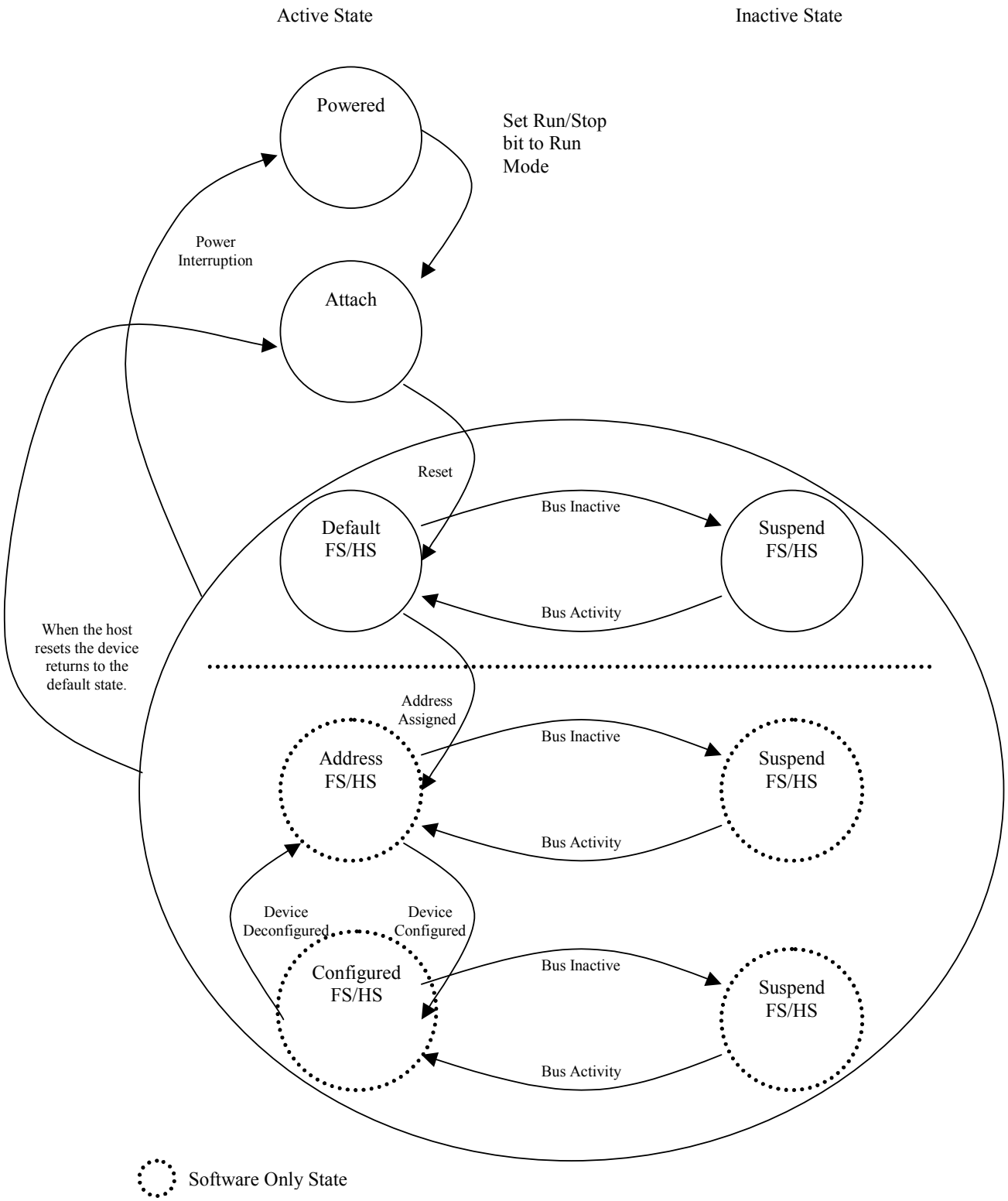


Figure 24-63. USB 2.0 Device States

States powered, attach, defaultFS/HS, suspendFS/HS are implemented in the USB_DR and are communicated to the DCD using the status bits identified in Table 24-79.

Table 24-80. Device Controller State Information Bits

Bit	Register
DCSuspend	USBSTS
USB Reset Received	USBSTS
Port Change Detect	USBSTS
High-Speed Port	PORTSC

It is the responsibility of the DCD to maintain a state variable to differentiate between the DefaultFS/HS state and the Address/Configured states. Change of state from Default to Address and the Configured states is part of the enumeration process described in the USB 2.0 Specification, Chapter 9, *Device Framework*.

As a result of entering the Address state, the device address register (DEVICEADDR) must be programmed by the DCD.

Entry into the Configured indicates that all endpoints to be used in the operation of the device have been properly initialized by programming the ENDPTCTRL*n* registers and initializing the associated queue heads.

24.11.2.1 Bus Reset

A bus reset is used by the host to initialize downstream devices. When a bus reset is detected, the USB_controller will renegotiate its attachment speed, reset the device address to 0, and notify 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 any primed transactions will be cancelled by the device controller. The concept of priming will be clarified below, but the DCD must perform the following tasks when a reset is received:

- Clear all setup token semaphores by reading the ENDPTSETUPSTAT register and writing the same value back to the ENDPTSETUPSTAT register.
- Clear all the endpoint complete status bits by reading the ENDPTCOMPLETE register and writing the same value back to the ENDPTCOMPLETE register.
- Cancel all primed status by waiting until all bits in the ENDPTPRIME are 0 and then writing 0xFFFF_FFFF to ENDPTFLUSH.

Read the reset bit in the PORTSC*n* register and make sure that it is still active. A USB reset will occur for a minimum of 3 ms and the DCD must reach this point in the reset cleanup before end of the reset occurs, otherwise a hardware reset of the device controller is recommended (rare.)

- A hardware reset can be performed by writing a one to the USB_DR reset bit in the USBCMD reset. Note: a hardware reset will cause the device to detach from the bus by clearing the Run/Stop bit. Thus, the DCD must completely re-initialize the USB_DR after a hardware reset.

Free all allocated dTDs because they will no longer be executed by the device controller. If this is the first time the DCD is processing a USB reset event, then it is likely that no dTDs have been allocated.

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.

After a Port Change Detect, the device has reached the default state and the DCD can read the PORTSC*n* to determine if the device is operating in FS or HS mode. At this time, the device controller has reached normal operating mode and DCD can begin enumeration according to the USB 2.0 Specification, Chapter 9, *Device Framework*.

NOTE

The device DCD may use the FS/HS mode information to determine the bandwidth mode of the device.

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.

24.11.2.2 Suspend/Resume

24.11.2.2.1 Suspend Description

In order to conserve power, USB_DR automatically enters the suspended state when no bus traffic has been observed for a specified period. When suspended, the USB_DR maintains any internal status, including its address and configuration. Attached devices must be prepared to suspend at any time they are powered, regardless of if they have been 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_DR 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_DR is capable of remote wake-up signaling. When the USB_DR is reset, remote wake-up signaling must be disabled.

24.11.2.2.2 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, the DCD is notified by an interrupt (assuming DC Suspend Interrupt is enabled). When the DCSuspend bit in the PORTSC*n* is set to a '1', the device controller is suspended.

DCD response when the device controller is suspended is application specific and may involve switching to low power operation.

Information on the bus power limits in suspend state can be found in USB 2.0 specification.

24.11.2.2.3 Resume

If the USB_DR is suspended, its operation is resumed when any non-idle signaling is received on its upstream facing port. In addition, the USB_DR can signal the system to resume operation by forcing resume signaling to the upstream port. Resume signaling is sent upstream by writing a '1' to the Resume bit in the in the PORTSC*n* while the device is in suspend state. 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 resume signaling can be used, the host must enable it by using the Set Feature command defined in USB 2.0 Specification, Chapter 9, *Device Framework*.

24.11.3 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. The endpoint address is specified by the combination of the endpoint number and the endpoint direction.

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 support by USB include bulk, interrupt, and isochronous. Each endpoint type has specific behavior related to packet response and error handling. More detail on endpoint operation can be found in the USB 2.0 specification.

The USB_DR supports up to six(6) endpoint specified numbers. The DCD can enable, disable and configure each endpoint.

Each endpoint direction is essentially independent and can be configured with 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 is, for example, is always a control endpoint and uses the pair of directions.

Each endpoint direction requires a queue head allocated in memory. If the maximum of 6 endpoint numbers, one for each endpoint direction are being used by the device controller, then 12 queue heads are required. The operation of an endpoint and use of queue heads are described later in this document.

24.11.3.1 Endpoint Initialization

After a hardware reset, all endpoints except endpoint zero are uninitialized and disabled. The DCD must configure and enable each endpoint by writing to configuration bit in the ENDPTCTRL*n* register. Each 32-bit ENDPTCTRL*n* is split into an upper and lower half. The lower half of ENDPTCTRL*n* is used to configure the receive or OUT endpoint and the upper half is likewise used to configure the corresponding transmit or IN endpoint. Control endpoints must be configured the same in both the upper and lower half of the ENDPTCTRL*n* register otherwise the behavior is undefined. Table 24-81 shows how to construct a configuration word for endpoint initialization.

Table 24-81. Device Controller Endpoint Initialization

Field	Value
Data Toggle Reset	1
Data Toggle Inhibit	0

Table 24-81. Device Controller Endpoint Initialization (continued)

Field	Value
Endpoint Type	00 Control 01 Isochronous 10 Bulk 11 Interrupt
Endpoint Stall	0

24.11.3.1.1 Stalling

There are two occasions where the USB_DR may need to return to the host a STALL.

The first occasion is the functional stall, which is a condition set by the DCD as described in the USB 2.0 Specification, Chapter 9, *Device Framework*. A functional stall is only used on non-control endpoints and can be enabled in the device controller by setting the endpoint stall bit in the ENDPTCTRL*n* register associated with the given endpoint and the given direction. In a functional stall condition, the device controller will continue 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 is automatically cleared by the device controller at the start of a new control transaction (setup phase). When enabling a protocol stall, the DCD should enable the stall bits (both directions) as a pair. A single write to the ENDPTCTRL*n* register can ensure that both stall bits are set at the same instant.

NOTE

Any write to the ENDPTCTRL*n* register during operational mode must preserve the endpoint type field (that is, perform a read-modify-write).

Table 24-82. 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

24.11.3.2 Data Toggle

Data toggle is a mechanism to maintain data coherency between host and device for any given data pipe. For more information on data toggle, refer to the USB 2.0 specification.

24.11.3.2.1 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 writing a '1' to the data toggle reset bit in the ENDPTCTRL*n* register. This should only be necessary when configuring/initializing an endpoint or returning from a STALL condition.

24.11.3.2.2 Data Toggle Inhibit

This feature is for test purposes only and should never be used during normal device controller operation.

Setting the data toggle Inhibit bit active ('1') causes the USB_DR to ignore the data toggle pattern that is normally sent and accept all incoming data packets regardless of the data toggle state.

In normal operation, the USB_DR checks the DATA0/DATA1 bit against the data toggle to determine if the packet is valid. If 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_DR from re-sending the same packet, the device controller will respond to the error packet by acknowledging it with either an ACK or NYET response.

24.11.3.3 Device Operational Model For Packet Transfers

All transactions on the USB bus are initiated by the host 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 will send requests to the USB_DR 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, then we can expect the host will send IN requests to that endpoint. This USB_DR 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 will be used throughout the following documentation to describe the USB_DR operation so the DCD can be architected properly use priming. Further, note that the term ‘flushing’ is used to describe the action of clearing a packet that was queued for execution.

24.11.3.3.1 Priming Transmit Endpoints

Priming a transmit endpoint will cause 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 will be 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 handle 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 is split into virtual channels so that 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 ENDPTSTATUS 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.

Since only the leading data is stored in the device controller FIFO, it is necessary for the device controller to begin filling in behind leading data after the transaction starts. The FIFO must be sized to account for the maximum latency that can be incurred by the system memory bus.

24.11.3.3.2 Priming Receive Endpoints

Priming receive endpoints is identical to priming of transmit endpoints from the point of view of the DCD. At the device controller the major difference in the operational model is that there is no data movement of the leading packet data simply because the data is to be received from the host.

Note 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.

24.11.3.4 Interrupt/Bulk Endpoint Operational Model

The behaviors of the device controller for interrupt and bulk endpoints are identical. All valid IN and OUT transactions to bulk pipes will handshake with a NAK unless the endpoint had been primed. Once the endpoint has been primed, data delivery will commence.

A dTD will be retired by the device controller when the packets described in the transfer descriptor have been completed. Each dTD describes N packets to be transferred according to the USB Variable Length transfer protocol. The Equation and Equation and Table 24-83 and Table 24-84 describe how the device controller computes the number and length of the packets to be sent/received by the USB vary according to the total number of bytes and maximum packet length.

With Zero Length Termination (ZLT) = 0

$$N = \text{INT}(\text{Number of Bytes}/\text{Maximum Packet Length}) + 1 \qquad \textit{Eqn. 24-1}$$

With Zero Length Termination (ZLT) = 1

$$N = \text{MAXINT}(\text{Number of Bytes}/\text{Maximum Packet Length}) \qquad \textit{Eqn. 24-2}$$

Table 24-83. 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 24-84. 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 dTD were successfully transmitted. *** Total bytes in dTD will equal zero when this occurs.

RX-dTD is complete when:

- All packets described in dTD were successfully received. *** Total bytes in dTD will equal zero when this occurs.
- A short packet (number of bytes < maximum packet length) was received. *** This is a successful transfer completion; DCD must check Total Bytes in dTD to determine the number of bytes that are 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 will discard the remaining packet, and set the Buffer Error bit in the dTD. In addition, the endpoint will be flushed and the USBERR interrupt will become active.

On the successful completion of the packet(s) described by the dTD, the active bit in the dTD will be cleared and the next pointer will be followed when the Terminate bit is clear. When the Terminate bit is set, the USB_DR will flush the endpoint/direction and cease operations for that endpoint/direction.

On the unsuccessful completion of a packet (see long packet above), the dQH will be left pointing to the dTD that was in error. In order to recover from this error condition, the 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 will be retried automatically by the device controller.

There is no required interaction with the DCD for handling such errors.

24.11.3.4.1 Interrupt/Bulk Endpoint Bus Response Matrix

Table 24-85. Interrupt/Bulk Endpoint Bus Response Matrix

	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.

SYSERR—System error should never occur when the latency FIFOs are correctly sized and the DCD is responsive.

24.11.3.5 Control Endpoint Operation Model

24.11.3.5.1 Setup Phase

All requests to a control endpoint begin with a setup phase followed by an optional data phase and a required status phase. The USB_DR will always accept the setup phase unless the setup lockout is engaged.

The setup lockout will engage so that future setup packets are ignored. Lockout of setup packets ensures that while the software is reading the setup packet stored in the queue head, that data is not written as it is being read potentially causing an invalid setup packet.

The setup lockout mechanism can be disabled and a tripwire type semaphore will ensure that the setup packet payload is extracted from the queue head without being corrupted be an incoming setup packet. This is the preferred behavior because ignoring repeated setup packets due to long software interrupt latency would be a compliance issue.

Setup Packet Handling

- Disable Setup Lockout by writing '1' to Setup Lockout Mode (SLOM) in USBMODE. (once at initialization). Setup lockout is not necessary when using the tripwire as described below.

NOTE

Leaving the Setup Lockout Mode As '0' will result in a potential compliance issue.

- After receiving an interrupt and inspecting ENDPTSETUPSTAT to determine that a setup packet was received on a particular pipe:
  - Write '1' to clear corresponding bit ENDPTSETUPSTAT.
  - Write '1' to Setup Tripwire (SUTW) in USBCMD register.
  - Duplicate contents of dQH.SetupBuffer into the local software byte array.
  - Read Setup TripWire (SUTW) in USBCMD register. (if set—continue; if cleared—goto 2)
  - Write '0' to clear Setup Tripwire (SUTW) in USBCMD register.
  - Process setup packet using the local software byte array copy and execute status/handshake phases.

  Note: After receiving a new setup packet the status and/or handshake phases may still be 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.

24.11.3.5.2 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 has not been received by reading the ENDPTSETUPSTAT register immediately verifying that the prime had completed. A prime will complete when the associated bit in the ENDPTPRIME register is zero and the associated bit in the ENDPTSTATUS register is a one. If a prime fails, that is, The ENDPTPRIME bit goes to zero and the ENDPTSTATUS bit

is not set, then the prime has failed. This can only be due to improper setup of the dQH, dTD or a setup arriving during the prime operation. If a new setup packet is indicated after the ENDPTPRIME bit is cleared, then the transfer descriptor can be freed and the DCD must reinterpret the setup packet.

Should a setup arrive after the data stage is primed, the device controller will automatically clear the prime status (ENDPTSTATUS) to enforce data coherency with the setup packet.

NOTE

The MULT field in the dQH must be set to ‘00’ for bulk, interrupt, and control endpoints.

NOTE

Error handling of data phase packets is the same as bulk packets described previously.

24.11.3.5.3 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 ENDPTSETUPSTAT as described above in the data phase.

NOTE

The MULT field in the dQH must be set to ‘00’ for bulk, interrupt, and control endpoints.

NOTE

Error handling of data phase packets is the same as bulk packets described previously.

24.11.3.5.4 Control Endpoint Bus Response Matrix

Table 24-86 shows the device controller response to packets on a control endpoint according to the device controller state.

Table 24-86. 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 should 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.

24.11.3.6 Isochronous Endpoint Operational Model

Isochronous endpoints are 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_DR will is accomplished by the following:

- Exactly MULT Packets per (micro)Frame are transmitted/received. Note: MULT is a two-bit field in the device Queue Head. The variable length packet protocol is not used on isochronous endpoints.
- NAK responses are not used. Instead, zero length packets and sent in response to an IN request to an 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 the ISO-dTD is still active after that frame, then the ISO-dTD will be held ready until executed or canceled by the DCD.

The USB_DR 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 handling 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 will become primed only after a SOF is received. After the DCD writes the prime bit, the prime bit will be cleared as usual to indicate to the 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 that 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. Once an ISO transaction is started in a (micro)frame it will retire 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, the device controller will force retire the ISO-dTD and move to the next ISO-dTD.

It is important to note that fulfillment errors are only caused due to partially completed packets. If no activity occurs to a primed ISO-dTD, the transaction will stay primed indefinitely. This means it is up to the software 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 handling. 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 is zero, the MULT Counter is initialized 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 the DCD should ensure that the dTD linked-list is out ahead of the device controller by at least two (micro)frames.

24.11.3.6.1 Isochronous Pipe Synchronization

When it is necessary to synchronize an isochronous data pipe to the host, the (micro)frame number (FRINDEX register) can be used as a marker. To cause a packet transfer to occur at a specific (micro)frame number [N], the DCD should interrupt on SOF during frame N-1. When the FRINDEX=N-1, the DCD must write the prime bit. The USB_DR will prime the isochronous endpoint in (micro)frame N-1 so that the device controller will execute delivery during (micro)frame N.

CAUTION

Priming an endpoint towards the end of (micro)frame N-1 will not guarantee delivery in (micro)frame N. The delivery may actually occur in (micro)frame N+1 if device controller does not have enough time to complete the prime before the SOF for packet N is received.

24.11.3.6.2 Isochronous Endpoint Bus Response Matrix

Table 24-87. Isochronous Endpoint Bus Response Matrix

	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
Out	Ignore	Ignore	Receive	N/A	Drop Packet

Table 24-87. Isochronous Endpoint Bus Response Matrix (continued)

	Stall	Not Primed	Primed	Underflow	Overflow
Ping	Ignore	Ignore	Ignore	Ignore	Ignore
Invalid	Ignore	Ignore	Ignore	Ignore	Ignore

1 Zero Length Packet
2 Force Bit Stuff Error

24.11.4 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 ENDPOINTLISTADDR contains a group of all dQH's in a sequential list as shown in Figure 24-64. The even elements in the list of dQH's are used for receive endpoints (OUT/SETUP) and the odd elements are used for transmit endpoints (IN/INTERRUPT). Device transfer descriptors are linked head to tail starting at the queue head and ending at a terminate bit. Once the dTD has been retired, it will no longer be part of the linked list from the queue head. Therefore, the software is required to track all transfer descriptors since pointers will no longer exist within the queue head once the dTD is retired (see Section 24.11.5.1, “Software Link Pointers”).

In addition to the current and next pointers and the dTD overlay examined in Section 24.11.3.3, “Device Operational Model For Packet Transfers”, the dQH also contains the following parameters for the associated endpoint: Multipler, Maximum Packet Length, Interrupt On Setup. The complete initialization of the dQH including these fields is demonstrated in the next section.

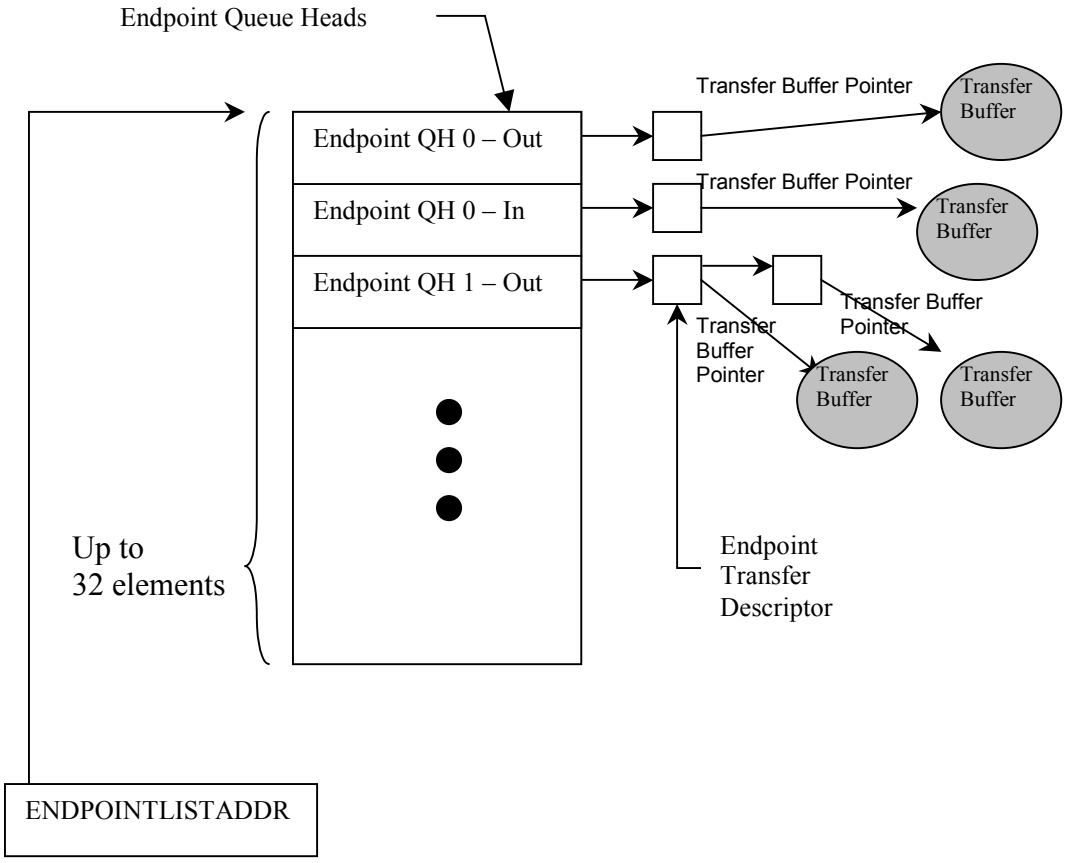


Figure 24-64. Endpoint Queue Head Diagram

24.11.4.1 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 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 bandwidth an in conjunction with the USB Chapter 9 protocol. Note: In FS mode, the multiplier field can only be 1 for ISO endpoints.
- Write the next dTD Terminate bit field to ‘1.’
- Write the Active bit in the status field to ‘0.’
- Write the Halt bit in the status field to ‘0.’

NOTE

The DCD must only modify dQH if the associated endpoint is not primed and there are no outstanding dTD's.

24.11.4.2 Operational Model For Setup Transfers

As discussed in Section 24.11.3.5, “Control Endpoint Operation Model,” setup transfer requires 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 handle the setup transfer as demonstrated here:

1. Copy setup buffer contents from dQH—RX to the software buffer.
2. Acknowledge setup backup by writing a 1 to the corresponding bit in ENDPTSETUPSTAT.

NOTE

The acknowledge must occur before continuing to process the setup packet.

NOTE

After the acknowledge has occurred, the DCD must not attempt to access the setup buffer in the dQH—RX. Only the local software copy should be examined.

3. Check for pending data or status dTD's from previous control transfers and flush if any exist as discussed in Section 24.11.5.5, “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. Decode setup packet and prepare data phase [optional] and status phase transfer as required by the USB Chapter 9 or application specific protocol.

24.11.5 Managing Transfers with Transfer Descriptors

24.11.5.1 Software Link Pointers

It is necessary for the DCD software to maintain head and tail pointers to the 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 to be executed. The operations described in next section for managing dTD will assume the DCD can use 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 it still remains the responsibility of the DCD to maintain the pointers.

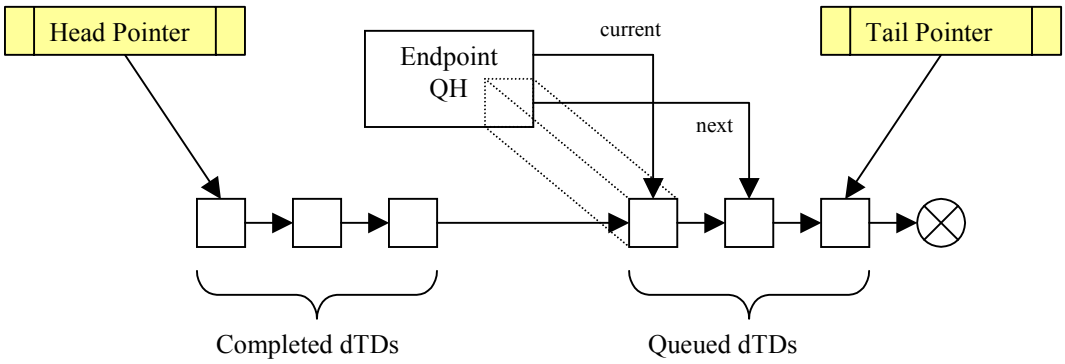


Figure 24-65. Software Link Pointers

24.11.5.2 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 8-DWord dTD block of memory aligned to 8-DWord boundaries. Example: bit address 4–0 would be equal to ‘00000’.

Write the following fields:

1. Initialize first 7 DWords to 0.
2. Set the terminate bit to ‘1.’
3. Fill in total bytes with transfer size.
4. Set the interrupt on complete if desired.
5. Initialize the status field with the active bit set to ‘1’ and all remaining status bits set to ‘0.’
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 pointer.

24.11.5.3 Executing A Transfer Descriptor

To safely add a dTD, the DCD must be follow this procedure which will handle the event where the device controller reaches the end of the dTD list at the same time a new dTD is being added to the end of the list.

Determine whether the link list is empty:

Check DCD driver to see if pipe is empty (internal representation of 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 DWord operation.
2. Clear active and halt bit in dQH (in case set from a previous error).
3. Prime endpoint by writing '1' to correct bit position in ENDPTPRIME.

Case 2: Link list is not empty

1. Add dTD to end of linked list.
2. Read correct prime bit in ENDPTPRIME—if 1 DONE.
3. Set ATDTW bit in USBCMD register to 1.
4. Read correct status bit in ENDPTSTATUS. (store in tmp. variable for later)
5. Read ATDTW bit in USBCMD register.
   If 0 goto 3.
   If 1 continue to 6.
6. Write ATDTW bit in USBCMD register to '0'.
7. If status bit read in (3) is '1' DONE.
8. If status bit read in (3) is '0' then Goto Case 1: Step 1.

24.11.5.4 Transfer Completion

After a dTD has been initialized and the associated endpoint primed the device controller will execute the transfer upon the host-initiated request. The DCD will be notified with a USB interrupt if the Interrupt On Complete bit was set or alternately, the DCD can poll the endpoint complete register to find when the dTD had been executed. After a dTD has been executed, DCD can check the status bits to determine success or failure.

CAUTION

Multiple dTD can be completed in a single endpoint complete notification. After clearing the notification, DCD must search the dTD linked list and retire all dTDs that have finished (Active bit cleared).

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 the 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 is by decremented by the actual bytes transferred. For Transmit packets, a packet is only complete after the actual bytes reaches zero, but for receive packets, the host may send fewer bytes in the transfer according the USB variable length packet protocol.

24.11.5.5 Flushing/De-Priming an Endpoint

It is necessary for the DCD to flush to de-prime one more endpoints on a USB device reset or during a broken control transfer. There may also be application specific requirements to stop transfers in progress. The following procedure can be used by the DCD to stop a transfer in progress:

1. Write a '1' to the corresponding bit(s) in ENDPTFLUSH.
2. Wait until all bits in ENDPTFLUSH are '0'.
3. Software 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.
4. Read ENDPTSTATUS to ensure that for all endpoints commanded to be flushed, that the corresponding bits are now '0'. If the corresponding bits are '1' after step #2 has finished, then the flush failed as described in the following:

   Explanation: In very rare cases, a packet is in progress to the particular endpoint when commanded flush using ENDPTFLUSH. 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 be repeating steps 1-3 until each endpoint is successfully flushed.

24.11.5.6 Device Error Matrix

Table 24-88 summarizes packet errors that are not automatically handled by the USB_DR.

Table 24-88. Device Error Matrix

Error	Direction	Packet Type	Data Buffer Error Bit	Transaction Error Bit
Overflow **	RX	Any	1	0
ISO Packet Error	RX	ISO	0	1
ISO Fulfillment Error	Both	ISO	0	1

Notice that the device controller handles 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 24-89. Error Descriptions

Error	Description
Overflow	Number of bytes received exceeded max. packet size or total buffer length. ** This error will also set the Halt bit in the dQH and if there are dTDs remaining in the linked list for the endpoint, then those will not be executed.
ISO Packet Error	CRC Error on received ISO packet. Contents not guaranteed to be 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 the DCD may need to readjust the data queue because a fulfillment error will cause 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.

24.11.6 Servicing Interrupts

The interrupt service routine must consider that there are high-frequency, low-frequency operations, and error operations and order accordingly.

24.11.6.1 High-Frequency Interrupts

High frequency interrupts in particular should be handed 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 24-90. Interrupt Handling Order

Execution Order	Interrupt	Action
1a	USB Interrupt[1] ENDPTSETUPSTATUS	Copy contents of setup buffer and acknowledge setup packet (as indicated in Section 24.11.4, “Managing Queue Heads”). Process setup packet according to USB 2.0 Chapter 9 or application specific protocol.
1b	USB Interrupt ENDPTCOMPLETE	Handle completion of dTD as indicated in Section 24.11.4, “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 that multiple interrupts to stack up on any call to the Interrupt Service Routine AND during the Interrupt Service Routine.

24.11.6.2 Low-Frequency Interrupts

The low frequency events include the following interrupts. These interrupt can be handled in any order since they don't occur often in comparison to the high-frequency interrupts.

Table 24-91. Low Frequency Interrupt Events

Interrupt	Action
Port Change	Change the software state information.
Sleep Enable (Suspend)	Change the software state information. Low power handling as necessary.
Reset Received	Change the software state information. Abort pending transfers.

24.11.6.3 Error Interrupts

Error interrupts will be least frequent and should be placed last in the interrupt service routine.

Table 24-92. 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 will more aptly handle packet-level errors by checking dTD status field upon receipt of USB Interrupt (w/ ENDPTCOMPLETE).
System Error	Unrecoverable error. Immediate Reset of core; free transfers buffers in progress and restart the DCD.

24.12 Deviations from the EHCI Specifications

The host mode operation of the modules is nearly EHCI-compatible with few minor differences. For the most part, the modules conform to the data structures and operations described in Section 3, “Data Structures,” and Section 4, “Operational Model,” in the EHCI specification. The particulars of the deviations occur in the following areas:

- Embedded Transaction Translator —Allows direct attachment of FS and LS devices in host mode without the need for a companion controller.
- Device operation —In host mode, the device operational registers are generally disabled and thus device mode is mostly transparent. However, there are a couple exceptions and they are documented in the following sections.
- Embedded design interface—The USB module does not have a PCI Interface and therefore the PCI configuration registers described in the EHCI specification are not applicable.

For the purposes of the USB OTG implementing dual-role host/device controller with support for OTG applications, it is necessary to deviate from the EHCI specification. Device operation and OTG operation are not specified in the EHCI and thus the implementation supported in the USB OTG module is proprietary.

24.12.1 Embedded Transaction Translator Function

In Host mode, the USB module 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 has been 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 that exist in the EHCI specification to support full and low speed devices.

24.12.1.1 Capability Registers

The following additions have been added to the capability registers to support the embedded Transaction Translator Function:

- N_TT added to HCSPARAMS—Host Controller Structural Parameters
- N_PTT added to HSCPARAMS—Host Controller Structural Parameters

See Section 24.6.2.3, “Host Controller Structural Parameters (HCSPARAMS)” for usage information.

24.12.1.2 Operational Registers

The addition of two-bit Port Speed (PSPD) to the PORTSCn register is added to the operational registers to support the embedded TT:

24.12.1.3 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 will only be set in a standard EHCI controller implementation after the port reset operation and when the host and device negotiate a High-Speed connection (that is, Chirp completes successfully).

The module will always set the port enable after the port reset operation regardless of the result of the host device chirp result and the resulting port speed will be indicated by the PSPD field in PORTSC*n*. Therefore, the standard EHCI host controller driver requires an alteration to handle directly connected Full and Low speed devices or hubs. The change is a fundamental one in that is summarized in Table 24-93.

Table 24-93. 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 thus, all port-level control is performed through the Hub Class to the nearest Hub.	FS and LS device can be either downstream from a HS hub or directly attached. When the FS/LS device is downstream from a HS hub, then port-level control is done 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=X. [where HubAddr > 0 and HubAddr is the address of the Hub where the bus transitions from HS to FS/LS (that is, Split target hub)]	FS and LS device can be either downstream from a HS hub with HubAddr = X [HubAddr > 0] or directly attached [where HubAddr = 0 and HubAddr is the address of the Root Hub where the bus transitions from HS to FS/LS (that is, Split target hub is the root hub)]

24.12.1.4 Data Structures

The same data structures used for FS/LS transactions though a HS hub are also used for transactions through the Root Hub. Here it is demonstrated how the Hub Address and Endpoint Speed fields should be set for directly attached FS/LS devices and hubs:

1. QH (for direct attach FS/LS) – Async. (Bulk/Control Endpoints) Periodic (Interrupt)

- Hub Address = 0
- Transactions to direct attached device/hub.
  - — QH.EPS = Port Speed
- Transactions to a device downstream from direct attached FS hub.
  - — QH.EPS = Downstream Device Speed

NOTE

When QH.EPS = 01 (LS) and PORTSC*n*[PSPD] = 00 (FS), a LS-pre-pid will be sent before the transmitting LS traffic.

Maximum Packet Size must be less than or equal 64 or undefined behavior may result.

2. siTD (for direct attach FS) – Periodic (ISO Endpoint)

- All FS ISO transactions:
  — Hub Address = 0
  — siTD.EPS = 00 (full speed)
  Maximum Packet Size must less than or equal to 1023 or undefined behavior may result.

24.12.1.5 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. Since the embedded Transaction Translator exists within the USB module there is no physical bus between EHCI host controller driver and the USB FS/LS bus. These sections will briefly discuss the operational model for how the EHCI and Transaction Translator operational models are combined without the physical bus between. The following sections assume the reader is familiar with both the EHCI and USB 2.0 Transaction Translator operational models.

24.12.1.5.1 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 shall use 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 will be ready to execute on the bus in B-frame 0.

It is important to note that 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.

Once periodic transfers are exhausted, any stored asynchronous transfer will be moved. Asynchronous transfers are opportunistic in that they shall execute whenever possible and their operation is not tied to H-frame and B-frame boundaries with the exception that an asynchronous transfer can not babble through the SOF (start of B-frame 0).

24.12.1.5.2 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 24-94 summarizes the conditions where handshakes are emulated from internal state instead of actual handshakes to HS split bus traffic.

Table 24-94. 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]

24.12.1.5.3 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-speed/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 Error! Reference source not found. register.

24.12.1.5.4 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 16 data pipes.
    - Some applications may not require transaction tracking up to a maximum of 16 periodic data pipes. The option to limit the tracking to only 4 periodic data pipes exists in the by changing the configuration constant VUSB_HS_TT_PERIODIC_CONTEXTS to 4. The result is a significant gate count savings to the core given the limitations implied.

CAUTION

Limiting the number of tracking pipes in the EMBedded—TT to four (4) will impose the restriction that no more than 4 periodic transactions (INTERRUPT/ISOCHRONOUS) can be scheduled through the embedded-tt per frame. The number 16 was chosen in the USB specification because it is sufficient to ensure that the high-speed to full-speed periodic pipeline can remain full. Keeping the pipeline full puts no constraint on the number of periodic transactions that can be scheduled in a frame and the only limit becomes the flight time of the packets on the bus.

— 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 msec) or else undefined behavior may result.

24.12.1.5.5 Multiple Transaction Translators

The maximum number of embedded Transaction Translators that is currently supported is one as indicated by the N_TT field in the HCSPARAMS register. See Section 24.6.2.3, “Host Controller Structural Parameters (HCSPARAMS),” for more information.

24.12.2 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 controller is initialized 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.

24.12.3 Non-Zero Fields the Register File

Some of the reserved fields and reserved addresses in the capability registers and operational registers have use in device mode, the following must be adhered to:

- 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).

24.12.4 SOF Interrupt

The SOF interrupt is a free running 125 µsec interrupt for host mode. EHCI does not specify this interrupt, but it has been added for convenience and as a potential software time base. Note that the free running interrupt is shared with the device-mode start-of-frame interrupt. See Section 24.6.3.2, “USB Status

Register (USBSTS)," and Section 24.6.3.3, "USB Interrupt Enable Register (USBINTR)," for more information.

24.12.5 Embedded Design

This is an Embedded USB Host Controller as defined by the EHCI specification and thus does not implement the PCI configuration registers.

24.12.5.1 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 µsec using the transceiver clock as a reference clock. That is, 60 MHz transceiver clock for 8-bit physical interfaces and full-speed serial interfaces or 30 MHz transceiver clock for 16-bit physical interfaces.

24.12.6 Miscellaneous Variations from EHCI

24.12.6.1 Programmable Physical Interface Behavior

The modules support multiple physical interfaces which 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 have been added to the PORTSC*n* register providing a capability that is not defined by the EHCI specification.

24.12.6.2 Discovery

24.12.6.2.1 Port Reset

The port connect methods specified by EHCI require setting the port reset bit in the register for a duration of 10 msec. Due to the complexity required to support the attachment of devices that are not high speed there are counter already present in the design that can count the 10 msec reset pulse to alleviate the requirement of the software to measure this duration. Therefore, the basic connection is then summarized as the following:

- [Port Change Interrupt] Port connect change occurs to notify the host controller driver that a device has attached.
- The software shall write a '1' to the reset the device.
- *The software shall write a '0' to the reset the device after 10 msec.*
  - This step, which is 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 will simple be ignored and the reset will continue until completion.
- [Port Change Interrupt] Port enable change occurs to notify the host controller that the device in now operational and at this point the port speed has been determined.

24.12.6.2.2 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 will re-assign 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 is read-only and always reads 0.
- A 2-bit Port Speed indicator in PORTSC register provides the current operating speed of the port to the host controller driver.

Chapter 25
FlexCAN Module

This chapter discusses the modes of operation, signals, memory map, register descriptions, and the functional and initialization sequence of the FlexCAN controller of the MCF5251.

25.1 Features

Following are the main features of the FlexCAN module:

- Full implementation of the CAN protocol specification version 2.0B
  - — Standard data and remote frames (up to 109 bits long)
  - — Extended data and remote frames (up to 127 bits long)
  - — 0–8 bytes data length
  - — Programmable bit rate up to 1 Mbps
  - — Content-related addressing
- Up to 32 flexible message buffers of zero to eight bytes data length, each configurable as Rx or Tx, all supporting standard and extended messages
- Listen-only mode capability
- Three programmable mask registers: global (for MBs 0–13 and 16–31), special for MB14, and special for MB15
- Programmable transmission priority scheme: lowest ID or lowest buffer number
- Time stamp based on 16-bit, free-running timer
- Global network time, synchronized by a specific message
- Programmable I/O modes
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is assumed)
- Open network architecture
- Multimaster bus
- High immunity to EMI
- Short latency time due to an arbitration scheme for high-priority messages

25.2 Block Diagram

A block diagram describing the various submodules of the FlexCAN module is shown in Figure 25-1. Each submodule is described in detail in subsequent sections. The message buffer architecture is shown in Figure 25-2.

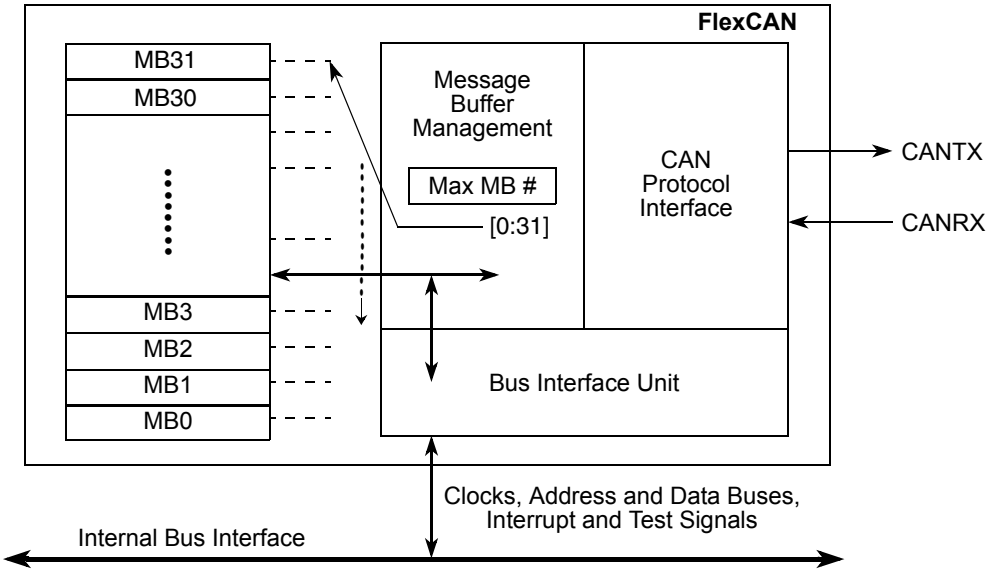


Figure 25-1. FlexCAN Block Diagram and Pinout

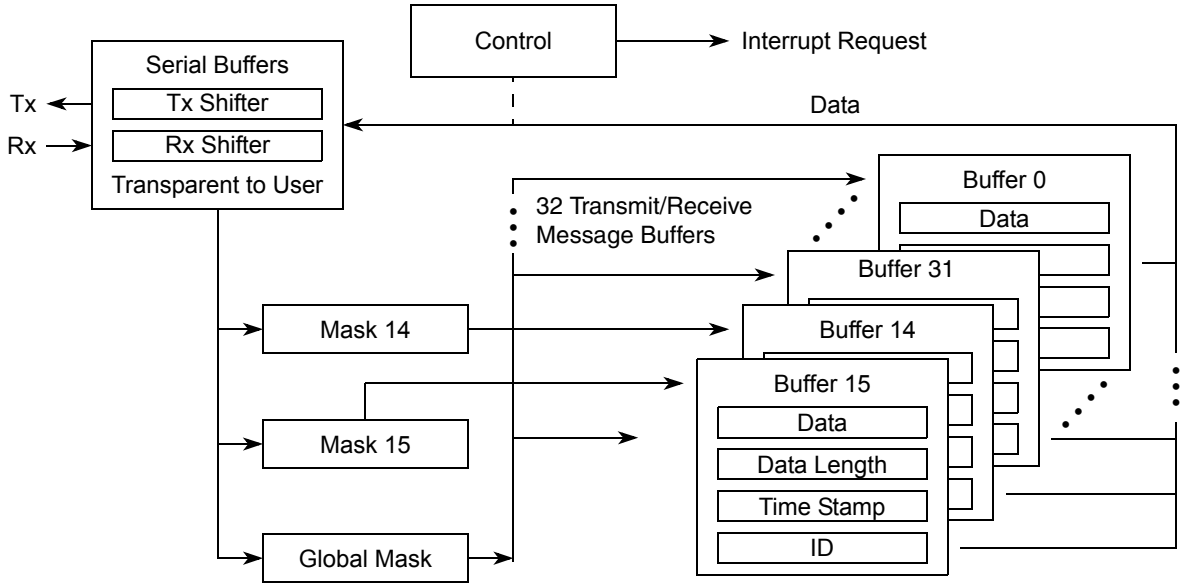


Figure 25-2. FlexCAN Message Buffer Architecture

25.3 Overview

The FlexCAN is a communication controller implementing the controller area network (CAN) protocol, an asynchronous communications protocol used in automotive and industrial control systems. It is a high speed (1 Mbps), short distance, priority-based protocol that can communicate using a variety of mediums (such as fiber optic cable or an unshielded twisted pair of wires). The FlexCAN supports both the standard and extended identifier (ID) message formats specified in the CAN protocol specification, revision 2.0, part B.

The CAN protocol was primarily, but not only, designed to be used as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness, and required bandwidth. A general working knowledge of the CAN protocol revision 2.0 is assumed in this document. For details, refer to the CAN protocol revision 2.0 specification.

25.3.1 The CAN System

A typical CAN system is shown in Figure 25-3.

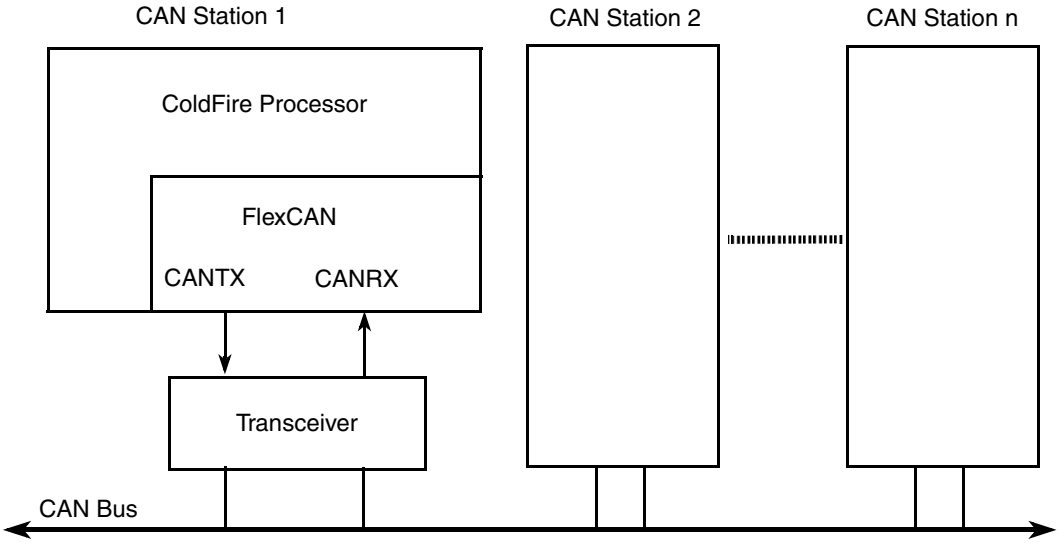


Figure 25-3. Typical CAN System

Each CAN station is connected physically to the CAN bus through a transceiver. The transceiver provides the transmit drive, waveshaping, and receive/compare functions required for communicating on the CAN bus. It can also provide protection against damage to the FlexCAN caused by a defective CAN bus or defective stations.

25.3.2 Modes of Operation

25.3.2.1 Normal Mode

In normal mode, the module operates receiving and/or transmitting message frames, errors are handled normally, and all the CAN protocol functions are enabled. User and supervisor modes differ in the access to some restricted control registers.

25.3.2.2 Freeze Mode

Freeze mode is entered by setting:

- CANMCR*n*[FRZ], and
- CANMCR*n*[HALT], or by asserting the $\overline{\text{BKPT}}$ signal.

Once entry into freeze mode is requested, the FlexCAN waits until an intermission or idle condition exists on the CAN bus, or until the FlexCAN enters the error passive or bus off state. Once one of these conditions exists, the FlexCAN waits for the completion of all internal activity such as arbitration, matching, move-in, and move-out. When this happens, the following events occur:

- The FlexCAN stops transmitting/receiving frames.
- The prescaler is disabled, thus halting all CAN bus communication.
- The FlexCAN ignores its Rx pins and drives its Tx pins as recessive.
- The FlexCAN loses synchronization with the CAN bus and the NOTRDY and FRZACK bits in CANMCR*n* are set.
- The CPU is allowed to read and write the error counter registers (in other modes they are read-only).

After engaging one of the mechanisms to place the FlexCAN in freeze mode, the user must wait for the FRZACK bit to be set before accessing any other registers in the FlexCAN; otherwise, unpredictable operation may occur. In freeze mode, all memory mapped registers are accessible.

To exit freeze mode, the $\overline{\text{BKPT}}$ line must be negated or the HALT bit in CANMCR*n* must be cleared. Once freeze mode is exited, the FlexCAN will resynchronize with the CAN bus by waiting for 11 consecutive recessive bits before beginning to participate in CAN bus communication.

25.3.2.3 Module Disabled Mode

This mode disables the FlexCAN module; it is entered by setting CANMCR*n*[MDIS]. If the module is disabled during freeze mode, it shuts down the system clocks, sets the LPMACK bit, and clears the FRZACK bit.

If the module is disabled during transmission or reception, FlexCAN does the following:

- Waits to be in either idle or bus-off state, or else waits for the third bit of intermission and then checks it to be recessive
- Waits for all internal activities such as arbitration, matching, move-in, and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Shuts down the system clocks

The bus interface unit continues to operate, enabling the CPU to access memory-mapped registers, except the free-running timer, the error counter register, and the message buffers, which cannot be accessed when the module is disabled. Exiting from this mode is done by negating the MDIS bit, which will resume the clocks and negate the LPMACK bit.

25.3.2.4 Loop-back Mode

The module enters this mode when the LPB bit in the control register is set. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is internally fed back to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic 1). FlexCAN behaves as it normally does when transmitting and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN

ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Both transmit and receive interrupts are generated.

25.3.2.5 Listen-only Mode

In listen-only mode, transmission is disabled, all error counters are frozen and the module operates in a CAN error passive mode. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message. Because the module does not influence the CAN bus in this mode, the device is capable of functioning like a monitor or for automatic bit-rate detection.

25.4 External Signal Description

Each FlexCAN module has two I/O signals connected to the external MPU pins: CAN0TX, CAN0RX, CAN1TX, and CAN1RX. CAN*n*TX transmits serial data to the CAN bus transceiver, while CAN*n*RX receives serial data from the CAN bus transceiver.

25.5 Memory Map and Register Definitions

The FlexCAN module address space is split into 128 bytes starting at the base address, and then an extra 512 bytes starting at the base address +128. The upper 512 are fully used for the message buffer structures, as described in Section 25.5.9, "Message Buffer Structure." Out of the lower 128 bytes, only part is occupied by various registers.

Table 25-1. FlexCAN Memory Map

MBAR2 Offset FlexCAN0 FlexCAN1	Register	Width (Bits)	Affected by Hard Reset	Affected by Soft Reset	Access	Reset Value	Section/Page
Supervisor-only Access Registers							
0x1000 0x2000	FlexCAN Module Configuration Register (CANMCR*n*)	32	Y	Y	R/W	0xD890_000F	25.5.1/25-6
Supervisor/User Access Registers							
0x1004 0x2004	FlexCAN Control Register (CANCTRL*n*)	32	Y	N	R/W	0x0000_0000	25.5.2/25-8
0x1008 0x2008	Free Running Timer (TIMER*n*)	32	Y	Y	R/W	0x0000_0000	25.5.3/25-11
0x1010 0x2010	Rx Global Mask (RXGMASK*n*)	32	Y	N	R/W	0x1FFF_FFFF	25.5.4/25-11
0x1014 0x2014	Rx Buffer 14 Mask (RX14MASk*n*)	32	Y	N	R/W	0x1FFF_FFFF	25.5.4/25-11
0x1018 0x2018	Rx Buffer 15 Mask (RX15MASK*n*)	32	Y	N	R/W	0x1FFF_FFFF	25.5.4/25-11

Table 25-1. FlexCAN Memory Map (continued)

MBAR2 Offset FlexCAN0 FlexCAN1	Register	Width (Bits)	Affected by Hard Reset	Affected by Soft Reset	Access	Reset Value	Section/Page
0x101C 0x201C	Error Counter Register (ERRCNT*n*)	32	Y	Y	R/W	0x0000_0000	25.5.6/25-14
0x1020 0x2020	Error and Status Register (ERRSTAT*n*)	32	Y	Y	R/W	0x0000_0000	25.5.6/25-14
0x1028 0x2028	Interrupt Mask Register (IMASK*n*)	32	Y	Y	R/W	0x0000_0000	25.5.7/25-16
0x1030 0x2030	Interrupt Flag Register (IFLAG*n*)	32	Y	Y	R/W	0x0000_0000	25.5.8/25-16
0x1080 0x2080	Message Buffers 0–31 (MB0–31)	4096	N	N	R/W	–	25.5.9/25-17
0x1180 0x2180	Message Buffers 16–31 (MB16–31)	2048	N	N	R/W	—	25.5.9/25-17

NOTE

The FlexCAN has no hard-wired protection against invalid bit/field programming within its registers. Specifically, no protection is provided if the programming does not meet CAN protocol requirements.

Programming the FlexCAN control registers is typically done during system initialization, prior to the FlexCAN becoming synchronized with the CAN bus. The configuration registers can be changed after synchronization by halting the FlexCAN module. This is done when the user sets the CANMCR*n*[HALT] bit. The FlexCAN responds by setting the CANMCR*n*[NOTRDY] bit.

25.5.1 FlexCAN Configuration Register (CANMCR*n*)

CANMCR*n* defines global system configurations, such as the module operation mode and maximum message buffer configuration. Most of the fields in this register can be accessed at any time, except the MAXMB field, which should only be changed while the module is in freeze mode.

Offset MBAR2 0x1000 (CANMCR0)
MBAR2 0x2000 (CANMCR1)

Access: Supervisor read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MDIS	FRZ	0	HALT	NOT RDY	0	SOFT RST	FRZ ACK	SUPV	0	0	LPM ACK	0	0	0	0
W																
Reset	1	1	0	1	1	0	0	0	1	0	0	1	0	0	0	0

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

Figure 25-4. FlexCAN Configuration Register (CANMCR*n*)

Table 25-2. FlexCAN Configuration Register (CANMCR*n*) Field Descriptions

Field	Description
31 MDIS	Module disable. This bit controls whether FlexCAN is enabled or not. When disabled, FlexCAN shuts down the FlexCAN clocks that drive the CAN interface and Message Buffer sub-module. This is the only bit in CANMCR*n* not affected by soft reset. See Section 25.3.2.3, "Module Disabled Mode," for more information. 0 Enable the FlexCAN module, clocks enabled 1 Disable the FlexCAN module, clocks disabled
30 FRZ	Freeze mode enable. When set, the FlexCAN can enter freeze mode when the $\overline{\text{BKPT}}$ line is asserted or the HALT bit is set. Clearing this bit causes the FlexCAN to exit freeze mode. Refer to Section 25.3.2.2, "Freeze Mode," for more information. 0 FlexCAN ignores the $\overline{\text{BKPT}}$ signal and the CANMCR*n*[HALT] bit. 1 FlexCAN module enabled to enter debug mode.
29	Reserved, should be cleared.
28 HALT	Halt FlexCAN. Setting this bit puts the FlexCAN module into freeze mode. It has the same effect as assertion of the $\overline{\text{BKPT}}$ signal. This bit is set after reset and should be cleared after initializing the message buffers and control registers. FlexCAN message buffer receive and transmit functions are inactive until this bit is cleared. While in freeze mode, the CPU has write access to the error counter register (ERRCNT*n*), that is otherwise read-only. 0 The FlexCAN operates normally 1 FlexCAN enters freeze mode if FRZ = 1
27 NOTRDY	FlexCAN not ready. This bit indicates that the FlexCAN is either in disable or freeze mode. This bit is read-only and it is cleared once the FlexCAN exits these modes. 0 FlexCAN is either in normal mode, listen-only mode, or loop-back mode. h1 FlexCAN is in disable or freeze mode.
26	Reserved, should be cleared.

Table 25-2. FlexCAN Configuration Register (CANMCR*n*) Field Descriptions (continued)

Field	Description
25 SOFTRST	Soft reset. When set, the FlexCAN resets its internal state machines (sequencer, error counters, error flags, and timer) and the host interface registers (CANMCR*n* [except the MDIS bit], TIMER, ERRCNT, ERRSTAT, IMASK, and IFLAG). The configuration registers that control the interface with the CAN bus are not changed (CANCTRL*n*, RXGMASK*n*, RX14MASK*n*, RX15MASK*n*). Message buffers are also not changed. This allows SOFTRST to be used as a debug feature while the system is running. Since soft reset is synchronous and has to follow a request/acknowledge procedure across clock domains, it may take some time to fully propagate its effect. The SOFTRST bit remains set while reset is pending and is automatically cleared when reset completes. The user should poll this bit to know when the soft reset has completed. 0 Soft reset cycle completed 1 Soft reset cycle initiated
24 FRZACK	Freeze acknowledge. Indicates that the FlexCAN module has entered freeze mode. The user should poll this bit after freeze mode has been requested, to know when the module has actually entered freeze mode. When freeze mode is exited, this bit is cleared once the FlexCAN prescaler is enabled. This is a read-only bit. 0 The FlexCAN has exited freeze mode and the prescaler is enabled. 1 The FlexCAN has entered freeze mode, and the prescaler is disabled.
23 SUPV	Supervisor/user data space. Places the FlexCAN registers in either supervisor or user data space. 0 Registers with access controlled by the SUPV bit are accessible in either user or supervisor privilege mode. 1 Registers with access controlled by the SUPV bit are restricted to supervisor mode.
22–21	Reserved, should be cleared.
20 LPMACK	Low power mode acknowledge. Indicates that FlexCAN is disabled. Disabled mode cannot be entered until all current transmission or reception processes have finished, so the CPU can poll the LPMACK bit to know when the FlexCAN has actually entered low power mode. See Section 25.3.2.3, “Module Disabled Mode” for more information. This bit is read-only. 0 FlexCAN not disabled. 1 FlexCAN is in disabled mode.
19–5	Reserved, should be cleared.
5–0 MAXMB	Maximum number of message buffers. Defines the maximum number of message buffers that will take part in the matching and arbitration process. The reset value (0x1F) is equivalent to 32 message buffer (MB) configuration. This field should be changed only while the module is in freeze mode. **Note:** Maximum MBs in Use = MAXMB + 1

25.5.2 FlexCAN Control Register (CANCTRL*n*)

CANCTRL*n* is defined for specific FlexCAN control features related to the CAN bus, such as bit-rate, programmable sampling point within an Rx bit, loop back mode, listen-only mode, bus off recovery behavior, and interrupt enabling. It also determines the division factor for the clock prescaler. Most of the fields in this register should only be changed while the module is disabled or in freeze mode. Exceptions are the BOFFMSK, ERRMSK, and BOFFREC bits, which can be accessed at any time.

Offset MBAR2 0x1004 (CANCTRL0)
MBAR2 0x2004 (CANCTRL1)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	PRESDIV								RJW		PSEG1			PSEG2		
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	BOFF MSK	ERR MSK	CLK_ SRC	LPB					SMP	BOFF REC	TSYN	LBUF	LOM	PROPSEG		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-5. FlexCAN Control (CANCTRL*n*) Register

Table 25-3. FlexCAN Control (CANCTRL*n*) Register Field Descriptions

Field	Description
31–24 PRESDIV	Prescaler division factor. Defines the ratio between the clock source frequency (set by CLK_SRC bit) and the serial clock (S clock) frequency. The S clock period defines the time quantum of the CAN protocol. For the reset value, the S clock frequency is equal to the clock source frequency. The maximum value of this register is 0xFF, that gives a minimum S clock frequency equal to the clock source frequency divided by 256. For more information refer to Section 25.6.8, “Bit Timing.” $\text{S clock frequency} = \frac{\text{SYSCLK CRIN}}{\text{PRESDIV} + 1}$ ***Eqn. 25-1***
23–22 RJW	Resynchronization jump width. Defines the maximum number of time quanta (one time quantum is equal to the S clock period) that a bit time can be changed by one resynchronization. The valid programmable values are 0–3. Resync jump width = (RJW + 1) time quanta ***Eqn. 25-2***
21–19 PSEG1	Phase buffer segment 1. Defines the length of phase buffer segment 1 in the bit time. The valid programmable values are 0–7. Phase buffer segment 1 = (PSEG1 + 1) time quanta ***Eqn. 25-3***
18–16 PSEG2	Phase buffer segment 2. Defines the length of phase buffer segment 2 in the bit time. The valid programmable values are 1–7. Phase buffer segment 2 = (PSEG2 + 1) time quanta ***Eqn. 25-4***
15 BOFFMSK	Bus off interrupt mask. 0 Bus off interrupt disabled 1 Bus off interrupt enabled
14 ERRMSK	Error interrupt mask. 0 Error interrupt disabled 1 Error interrupt enabled
13 CLK_SRC	Clock source. Selects the clock source for the CAN interface to be fed to the prescalar. This bit should only be changed while the module is disabled. 0 Clock source is CRIN 1 Clock source is the internal bus clock, SYSCLK

Table 25-3. FlexCAN Control (CANCTRL*n*) Register Field Descriptions (continued)

Field	Description
12 LPB	Loop back. Configures FlexCAN to operate in loop-back mode. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is fed back internally to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic 1). FlexCAN behaves as it normally does when transmitting, and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field, generating an internal acknowledge bit to ensure proper reception of its own message. Both transmit and receive interrupts are generated. 0 Loop back disabled 1 Loop back enabled
11–8	Reserved, should be cleared.
7 SMP	Sampling mode. Determines whether the FlexCAN module will sample each received bit one time or three times to determine its value. 0 One sample, taken at the end of phase buffer segment 1, is used to determine the value of the received bit. 1 Three samples are used to determine the value of the received bit. The samples are taken at the normal sample point and at the two preceding periods of the S-clock; a majority rule is used.
6 BOFFREC	Bus off recovery mode. Defines how FlexCAN recovers from bus off state. If this bit is cleared, automatic recovering from bus off state occurs according to the *CAN Specification 2.0B*. If the bit is set, automatic recovering from bus off is disabled and the module remains in bus off state until the bit is cleared by the user. If the bit is cleared before 128 sequences of 11 recessive bits are detected on the CAN bus, then bus off recovery happens as if the BOFFREC bit had never been set. If the bit is cleared after 128 sequences of 11 recessive bits occurred, then FlexCAN will re-synchronize to the bus by waiting for 11 recessive bits before joining the bus. After clearing, the BOFFREC bit can be set again during bus off, but it will only be effective the next time the module enters bus off. If BOFFREC was cleared when the module entered bus off, setting it during bus off will not be effective for the current bus off recovery. 0 Automatic recovering from bus off state enabled, according to CAN Spec 2.0B 1 Automatic recovering from bus off state disabled
5 TSYN	Timer synchronize mode. Enables the mechanism that resets the free-running timer each time a message is received in Message Buffer 0. This feature provides the means to synchronize multiple FlexCAN stations with a special “SYNC” message (global network time). 0 Timer synchronization disabled. 1 Timer synchronization enabled. Note: There can be a bit clock skew of four to five counts between different FlexCAN modules that are using this feature on the same network.
4 LBUF	Lowest buffer transmitted first. Defines the ordering mechanism for message buffer transmission. 0 Message buffer with lowest ID is transmitted first 1 Lowest numbered buffer is transmitted first
3 LOM	Listen-only mode. Configures FlexCAN to operate in listen-only mode. In this mode transmission is disabled, all error counters are frozen, and the module operates in a CAN error passive mode. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message. 0 FlexCAN module is in normal active operation; listen-only mode is deactivated 1 FlexCAN module is in listen-only mode operation
2–0 PROPSEG	Propagation segment. Defines the length of the propagation segment in the bit time. The valid programmable values are 0–7. Propagation segment time = (PROPSEG + 1) time-quanta **Note:** A time-quantum = 1 S clock period.

25.5.3 FlexCAN Free Running Timer Register (TIMER*n*)

This register represents a 16-bit free running counter that can be read and written to by the CPU. The timer starts from 0x0000 after reset, counts linearly to 0xFFFF, and wraps around.

The timer is clocked by the FlexCAN bit-clock (which defines the baud rate on the CAN bus). During a message transmission/reception, it increments by one for each bit that is received or transmitted. When there is no message on the bus, it counts using the previously programmed baud rate. During freeze mode, the timer is not incremented.

The timer value is captured at the beginning of the identifier (ID) field of any frame on the CAN bus. This captured value is written into the TIMESTAMP entry in a message buffer after a successful reception or transmission of a message.

Writing to the timer is an indirect operation. The data is first written to an auxiliary register, then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

Offset MBAR2 0x1008 (TIMER0)
MBAR2 0x2008 (TIMER1)

Access: User read/write

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

Figure 25-6. FlexCAN Timer (TIMER*n*) Register

Table 25-4. FlexCAN Timer (TIMER*n*) Register Field Descriptions

Field	Description
31–16	Reserved, should be cleared.
15–0 TIMER	Free running timer. Captured at the beginning of the identifier (ID) field of any frame on the CAN bus. This captured value is written into the TIMESTAMP entry in a message buffer after a successful reception or transmission of a message.

25.5.4 Rx Mask Registers (RXGMASK*n*, RX14MASK*n*, RX15MASK*n*)

These registers are used as acceptance masks for received frame IDs. Three masks are defined: a global mask (RXGMASK*n*) used for Rx buffers 0–13, 16–31 and two separate masks for buffers 14 (RX14MASK*n*) and 15 (RX15MASK*n*). The meaning of each mask bit is the following:

MI*n* bit = 0: The corresponding incoming ID bit is “don’t care”.

MI*n* bit = 1: The corresponding ID bit is checked against the incoming ID bit, to see if a match exists.

Note that these masks are used both for standard and extended ID formats. The value of the mask registers should not be changed while in normal operation (only while in freeze mode), as locked frames that matched a message buffer (MB) through a mask may be transferred into the MB (upon release) but may no longer match.

Table 25-5. Mask Examples for Normal/Extended Messages

	Base ID ID28.....ID18	IDE	Extended ID ID17.....ID0	Match
MB2-ID	1 1 1 1 1 1 1 1 0 0 0	0		–
MB3-ID	1 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	–
MB4-ID	0 0 0 0 0 0 1 1 1 1 1	0	–	–
MB5-ID	0 0 0 0 0 0 1 1 1 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	–
MB14-ID	1 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	–
Rx_Global_Mask	1 1 1 1 1 1 1 1 1 1 0		1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1	
Rx_Msg in[1]	1 1 1 1 1 1 1 1 0 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	MB3[1]
Rx_Msg in[2]	1 1 1 1 1 1 1 1 0 0 1	0	–	MB2[2]
Rx_Msg in[3]	1 1 1 1 1 1 1 1 0 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0	[3]
Rx_Msg in[4]	0 1 1 1 1 1 1 1 0 0 0	0	–	[4]
Rx_Msg in[5]	0 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	MB14[5]
RX14MASK	0 1 1 1 1 1 1 1 1 1 1		1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0	
Rx_Msg in[6]	1 0 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	[6]
Rx_Msg in[7]	0 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	MB14[7]

[1] Match for Extended Format (MB3).

[2] Match for Normal Format. (MB2).

[3] Mismatch for MB3 because of ID0.

[4] Mismatch for MB2 because of ID28.

[5] Mismatch for MB3 because of ID28, Match for MB14 (Uses RX14MASK*n*).

[6] Mismatch for MB14 because of ID27 (Uses RX14MASK*n*).

[7] Match for MB14 (Uses RX14MASK*n*).

Offset MBAR2 0x1010 (RXGMASK0) Access: User read/write
MBAR2 0x1014 (RX14MASK0)
MBAR2 0x1018 (RX15MASK0)
MBAR2 0x2010 (RXGMASK1)
MBAR2 0x2014 (RX14MASK1)
MBAR2 0x1018 (RX15MASK1)

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

Figure 25-7. FlexCAN Rx Mask (RXGMASK*n*, RX14MASK*n*, RX15MASK*n*) Registers

Table 25-6. FlexCAN Rx Mask (RXGMASK*n*, RX14MASK*n*, RX15MASK*n*) Registers Field Descriptions

Field	Description
31–29	Reserved, should be cleared.
28–18 MI28–18	Standard ID mask bits. These bits are the same mask bits for the Standard and Extended Formats.
17–0 MI17–0	Extended ID mask bits. These bits are used to mask comparison only in Extended Format.

25.5.5 FlexCAN Error Counter Register (ERRCNT*n*)

This register has two 8-bit fields reflecting the value of two FlexCAN error counters: transmit error counter (TXECTR) and receive error counter (RXECTR). The rules for increasing and decreasing these counters are described in the CAN protocol and are completely implemented in the FlexCAN module. Both counters are read-only, except in freeze mode, where they can be written by the CPU.

Writing to the ERRCNT*n* register while in freeze mode is an indirect operation. The data is first written to an auxiliary register, then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

FlexCAN responds to any bus state as described in the protocol, e.g. transmit error-active or error-passive flag, delay its transmission start time (error-passive), and avoid any influence on the bus when in bus off state. The following are the basic rules for FlexCAN bus state transitions:

- If the value of TXECTR or RXECTR increases to be greater than or equal to 128, the FLTCONF field in the error and status register (ERRSTAT*n*) is updated to reflect error-passive state.
- If the FlexCAN state is error-passive, and either TXECTR or RXECTR decrements to a value less than or equal to 127 while the other already satisfies this condition, the ERRSTAT*n*[FLTCONF] field is updated to reflect error-active state.
- If the value of TXECTR increases to be greater than 255, the ERRSTAT*n*[FLTCONF] field is updated to reflect bus off state, and an interrupt may be issued. The value of TXECTR is then reset to zero.
- If FlexCAN is in bus off state, then TXECTR is cascaded together with another internal counter to count the 128th occurrences of 11 consecutive recessive bits on the bus. Hence, TXECTR is reset to zero and counts in a manner where the internal counter counts 11 such bits and then wraps around while incrementing the TXECTR. When TXECTR reaches the value of 128, the ERRSTAT*n*[FLTCONF] field is updated to be error-active, and both error counters are reset to zero. At any instance of a dominant bit following a stream of less than 11 consecutive recessive bits, the internal counter resets itself to zero without affecting the TXECTR value.
- If during system start-up, only one node is operating, then its TXECTR increases in each message it is trying to transmit, as a result of acknowledge errors (indicated by the ERRSTAT*n*[ACKERR] bit). After the transition to error-passive state, the TXECTR does not increment anymore by acknowledge errors. Therefore, the device never goes to the bus off state.

- If the RXECTR increases to a value greater than 127, it is not incremented further, even if more errors are detected while being a receiver. At the next successful message reception, the counter is set to a value between 119 and 127 to resume to error-active state.

Offset MBAR2 0x101C (ERRCNT0)
MBAR2 0x201C (ERRCNT1)

Access: User read/write

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

Figure 25-8. FlexCAN Error Counter (ERRCNT*n*) Register

Table 25-7. FlexCAN Error Counter (ERRCNT*n*) Register Field Descriptions

Field	Description
31–16	Reserved, should be cleared.
15–8 RXECTR	Receive error counter. Indicates current number of receive errors.
7–0 TXECTR	Transmit error counter. Indicates current number of transmit errors.

25.5.6 FlexCAN Error and Status Register (ERRSTAT*n*)

ERRSTAT*n* reflects various error conditions, some general status of the device, and is the source of three interrupts to the CPU. The reported error conditions (bits 15:10) are those occurred since the last time the CPU read this register. The read action clears bits 15-10. Bits 9–3 are status bits.

Most bits in this register are read only, except for BOFFINT, WAKINT, and ERRINT, which are interrupt flags that can be cleared by writing 1 to them. Writing 0 has no effect. Refer to Section 25.7.1, “Interrupts.”

Offset MBAR2 0x1020 (CANCTRL0)
MBAR2 0x2020 (CANCTRL1)

Access: User read/write

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BIT1 ERR	BIT0 ERR	ACK ERR	CRC ERR	FRM ERR	STF ERR	TX WRN	RX WRN	IDLE	TXRX	FLT CONF		0	BOFF INT	ERR INT	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-9. FlexCAN Error and Status (ERRSTAT*n*) Register

Table 25-8. FlexCAN Error and Status (ERRSTAT*n*) Register Field Descriptions

Field	Description
31–16	Reserved, should be cleared.
15 BIT1ERR	Bit1 error. Indicates inconsistency between the transmitted and received bit in a message. 0 No transmit bit error 1 At least one bit sent as recessive was received as dominant **Note:** The transmit bit error field is not modified during the arbitration field or the ACK slot bit time of a message, or by a transmitter that detects dominant bits while sending a passive error frame.
14 BIT0ERR	Bit0 error. Indicates inconsistency between the transmitted and received bit in a message. 0 No transmit bit error 1 At least one bit sent as dominant was received as recessive
13 ACKERR	Acknowledge error. Indicates whether an acknowledgment has been correctly received for a transmitted message. 0 No ACK error was detected since the last read of this register. 1 An ACK error was detected since the last read of this register.
12 CRCERR	Cyclic redundancy check error. Indicates whether or not a CRC error has been detected by the receiver. 0 No CRC error was detected since the last read of this register. 1 A CRC error was detected since the last read of this register.
11 FRMERR	Message form error. Indicates that a form error has been detected by the receiver node, i.e., a fixed-form bit field contains at least one illegal bit. 0 No form error was detected since the last read of this register. 1 A form error was detected since the last read of this register.
10 STFERR	Bit stuff error. 0 No bit stuffing error was detected since the last read of this register. 1 A bit stuffing error was detected since the last read of this register.
9 TXWRN	Transmit error status flag. Reflects the status of the FlexCAN transmit error counter. 0 Transmit error counter < 96 1 TXErrCounter ≥ 96
8 RXWRN	Receiver error status flag. Reflects the status of the FlexCAN receive error counter. 0 Receive error counter < 96 1 RxErrCounter ≥ 96
7 IDLE	Idle status. Indicates when there is activity on the CAN bus. 0 The CAN bus is not idle. 1 The CAN bus is idle.
6 TXRX	Transmit/receive status. Indicates when the FlexCAN module is transmitting or receiving a message. TXRX has no meaning when IDLE = 1. 0 The FlexCAN is receiving a message if IDLE = 0. 1 The FlexCAN is transmitting a message if IDLE = 0.
5–4 FLTCONF	Fault confinement state. Indicates the confinement state of the FlexCAN module, as shown below. If the CANCTRL*n*[LOM] bit is set, FLTCONF will indicate error-passive. Since the CANCTRL*n* register is not affected by soft reset, the FLTCONF field will not be affected by soft reset if the LOM bit is set. 00 Error active 01 Error passive 1x Bus off
3	Reserved, should be cleared.

Table 25-8. FlexCAN Error and Status (ERRSTAT*n*) Register Field Descriptions (continued)

Field	Description
2 BOFFINT	Bus off interrupt. Used to request an interrupt when the FlexCAN enters the bus off state. The user must write a 1 to clear this bit. Writing 0 has no effect. 0 No bus off interrupt requested. 1 This bit is set when the FlexCAN state changes to bus off. If the CANCTRL*n*[BOFFMSK] bit is set an interrupt request is generated. This interrupt is not requested after reset.
1 ERRINT	Error interrupt. Indicates that at least one of the ERRSTAT*n*[15:10] bits is set. The user must write a 1 to clear this bit. Writing 0 has no effect. 0 No error interrupt request. 1 At least one of the error bits is set. If the CANCTRL*n*[ERRMSK] bit is set, an interrupt request is generated.
0	Reserved, should be cleared.

25.5.7 Interrupt Mask Register (IMASK*n*)

IMASK*n* contains one interrupt mask bit per buffer. It enables the CPU to determine which buffer will generate an interrupt after a successful transmission/reception (that is, when the corresponding IFLAG*n* bit is set).

Offset MBAR2 0x102A (IMASK0) Access: User read/write

	31	30	29	28	27	26	25	24	23	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	BUF*n*M, *n*=31–0																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-10. FlexCAN Interrupt Mask (IMASK*n*) Register

Table 25-9. FlexCAN Interrupt Mask (IMASK*n*) Register Field Descriptions

Field	Description
31–0 BUF*n*M	Buffer interrupt mask. Enables the respective FlexCAN message buffer (MB0 to MB31) interrupt. These bits allow the CPU to designate which buffers will generate interrupts after successful transmission/reception. 0 The interrupt for the corresponding buffer is disabled. 1 The interrupt for the corresponding buffer is enabled. **Note:** Setting or clearing an IMASK*n* bit can assert or negate an interrupt request, if the corresponding IFLAG*n* bit it is set.

25.5.8 Interrupt Flag Register (IFLAG*n*)

IFLAG*n* contains one interrupt flag bit per buffer. Each successful transmission/reception sets the corresponding IFLAG*n* bit and, if the corresponding IMASK*n* bit is set, will generate an interrupt.

The interrupt flag is cleared by writing a 1, while writing 0 has no effect.

Address MBAR2 0x1030 (IFLAG0) Access: User read/write

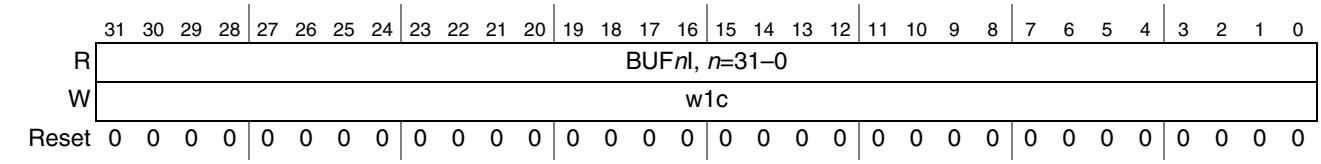


Figure 25-11. FlexCAN Interrupt Flags (IFLAG*n*) Register

Table 25-10. FlexCAN Interrupt Flags (IFLAG*n*) Register Field Descriptions

Field	Description
31–0 BUF*n*I	Buffer interrupt flag. Indicates a successful transmission/reception for the corresponding message buffer. If the corresponding IMASK*n* bit is set, an interrupt request will be generated. The user must write a 1 to clear an interrupt flag; writing 0 has no effect. 0 No such occurrence. 1 The corresponding buffer has successfully completed transmission or reception.

25.5.9 Message Buffer Structure

The message buffer memory map starts at an offset of 0x80 from the FlexCAN's base address 1 or CAN1: 0x2000). The 512-byte message buffer space is fully used by the 32 message buffer structures.

Each message buffer consists of a control and status field that configures the message buffer, an identifier field for frame identification, and up to 8 bytes of data.

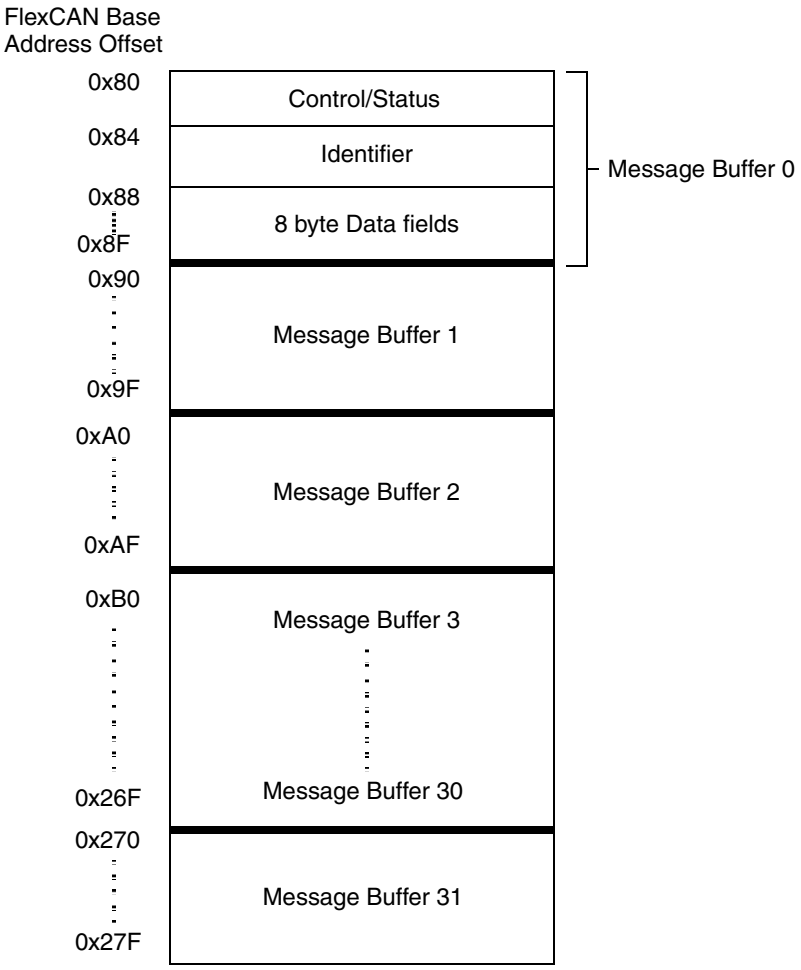


Figure 25-12. FlexCAN Message Buffer Memory Map

The message buffer structure used by the FlexCAN module is shown in Figure 25-13. Both standard and extended frames used in the *CAN Specification Version 2.0, Part B* are represented. A standard frame is represented by the 11-bit standard identifier, and an extended frame is represented by the combined 29-bits of the standard identifier (11 bits) and the extended identifier (18 bits).

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0					CODE					SRR	IDE	RTR	LENGTH				TIME STAMP															
0x4				Standard ID[28:18]													Extended ID[17:0]															
0x8	Data Byte 0								Data Byte 1								Data Byte 2								Data Byte 3							
0xC	Data Byte 4								Data Byte 5								Data Byte 6								Data Byte 7							

Figure 25-13. Message Buffer Structure Register for Both Extended and Standard Frames

Table 25-11. Message Buffer Structure Register Field Descriptions

Field	Description
31–28	Reserved, should be cleared.
27–24 CODE	Message buffer code. Can be accessed (read or write) by the CPU and by the FlexCAN module itself, as part of the message buffer matching and arbitration process. The encoding is shown in Table 25-12 and Table 25-13. See Section 25.6, “Functional Overview,” for additional information.
23	Reserved, should be cleared.
22 SRR	Substitute remote request. Fixed recessive bit, used only in extended format. It must be set by the user for transmission (Tx Buffers) and will be stored with the value received on the CAN bus for Rx receiving buffers. It can be received as either recessive or dominant. If FlexCAN receives this bit as dominant, then it is interpreted as arbitration loss. 0 Dominant is not a valid value for transmission in Extended Format frames 1 Recessive value is compulsory for transmission in Extended Format frames
21 IDE	ID extended bit. Identifies whether the frame format is standard or extended. 0 Standard frame format 1 Extended frame format
20 RTR	Remote transmission request. Used for requesting transmissions of a data frame. If FlexCAN transmits this bit as 1 (recessive) and receives it as 0 (dominant), it is interpreted as arbitration loss. If this bit is transmitted as 0 (dominant), then if it is received as 1 (recessive), the FlexCAN module treats it as bit error. If the value received matches the value transmitted, it is considered as a successful bit transmission. 0 Indicates the current MB has a data frame to be transmitted 1 Indicates the current MB has a remote frame to be transmitted
19–16 LENGTH	Length of data in bytes. Indicates the length (in bytes) of the Rx or Tx data; data is located in offset 0x8 through 0xF of the MB space (see Figure 25-13). In reception, this field is written by the FlexCAN module, copied from the DLC (data length code) field of the received frame. DLC is defined by the *CAN Specification* and refers to the data length of the actual frame before it is copied into the message buffer. In transmission, this field is written by the CPU and is used as the DLC field value of the frame to be transmitted. When RTR is set, the frame to be transmitted is a remote frame and will be transmitted without the DATA field, regardless of the LENGTH field.
15–-0 TIME STAMP	Free-running counter time stamp. Stores the value of the free-running timer which is captured when the beginning of the identifier (ID) field appears on the CAN bus.
31–29	Reserved, should be cleared.
28–0 ID	Standard frame identifier: In standard frame format, only the 11 most significant bits (28 to 18) are used for frame identification in both receive and transmit cases. The 18 least significant bits are ignored.
	Extended frame identifier: In extended frame format, all bits (both the 11 bits of the standard frame identifier and the 18 bits of the extended frame identifier) are used for frame identification in both receive and transmit cases.
31–24, 23–16, 15–8, 7–0 DATA	Data field. Up to eight bytes can be used for a data frame. For Rx frames, the data is stored as it is received from the CAN bus. For Tx frames, the CPU provides the data to be transmitted within the frame.

Table 25-12. Message Buffer Code for Rx Buffers

Rx Code BEFORE Rx New Frame	Description	Rx Code AFTER Rx New Frame	Comment
0000	INACTIVE: MB is not active	–	MB does not participate in the matching process.
0100	EMPTY: MB is active and empty	0010	MB participates in the matching process. When a frame is received successfully, the code is automatically updated to FULL.
0010	FULL: MB is full	0010	The act of reading the control & status (C/S) word followed by unlocking the MB does not make the code return to EMPTY. It remains FULL. If a new frame is written to the MB after the C/S word was read and the MB was unlocked, the code still remains FULL.
		0110	If the MB is FULL and a new frame should be written into this MB before the CPU had time to read it, the MB is overwritten, and the code is automatically updated to OVERRUN.
0110	OVERRUN: A frame was overwritten into a full buffer	0010	If the code indicates OVERRUN but the CPU reads the C/S word and then unlocks the MB, when a new frame is written to the MB, the code returns to FULL.
		0110	If the code already indicates OVERRUN, and yet another new frame must be written, the MB will be overwritten again, and the code will remain OVERRUN.
0XY1[1]	BUSY: Flexcan is updating the contents of the MB with a new receive frame. The CPU should not try to access the MB.	0010	An EMPTY buffer was written with a new frame (XY was 01).
		0110	A FULL/OVERRUN buffer was overwritten (XY was 11).

[1] Note that for transmit message buffers (see Table 25-13), the BUSY bit should be ignored upon read.

Table 25-13. Message Buffer Code for Tx Buffers

MB*n*[RTR]	Initial Tx Code	Code After Successful Transmission	Description
X	1000	–	INACTIVE: Message buffer not ready for transmit and will participate in the arbitration process.
0	1100	1000	Data frame to be transmitted once, unconditionally. After transmission, the MB automatically returns to the INACTIVE state.
1	1100	0100	Remote frame to be transmitted unconditionally once, and message buffer becomes an Rx message buffer with the same ID for data frames.

Table 25-13. Message Buffer Code for Tx Buffers (continued)

MB*n*[RTR]	Initial Tx Code	Code After Successful Transmission	Description
0	1010	1010	Transmit a data frame whenever a remote request frame with the same ID is received. This message buffer participates simultaneously in both the matching and arbitration processes. The matching process compares the ID of the incoming remote request frame with the ID of the MB. If a match occurs, this message buffer is allowed to participate in the current arbitration process and the CODE field is automatically updated to 1110 to allow the MB to participate in future arbitration runs. When the frame is eventually transmitted successfully, the code automatically returns to 1010 to restart the process again.
0	1110	1010	This is an intermediate code that is automatically written to the message buffer as a result of match to a remote request frame. The data frame will be transmitted unconditionally once, and then the code will automatically return to 1010. The CPU can also write this code with the same effect.

25.6 Functional Overview

The FlexCAN module is flexible in that each one of its 32 message buffers (MBs) can be assigned either as a transmit buffer or a receive buffer. Each MB, which is up to 8 bytes long, is also assigned an interrupt flag bit that indicates successful completion of either transmission or reception.

An arbitration algorithm decides the prioritization of MBs to be transmitted based on either the message ID or the MB ordering. A matching algorithm makes it possible to store received frames only into MBs that have the same ID programmed on its ID field. A masking scheme makes it possible to match the ID programmed on the MB with a range of IDs on received CAN frames. Data coherency mechanisms are implemented to guarantee data integrity during MB manipulation by the CPU.

Before proceeding with the functional description, an important concept must be explained. A message buffer is said to be active at a given time if it can participate in the matching and arbitration algorithms that are happening at that time. An Rx MB with a 0000 code is inactive (refer to Table 25-12). Similarly, a Tx MB with a 1000 code is inactive (refer to Table 25-13). An MB not programmed with either 0000 or 1000 will be temporarily deactivated (will not participate in the current arbitration/matching run) when the CPU writes to the C/S field of that MB.

25.6.1 Transmit Process

The CPU prepares or changes an MB for transmission by writing the following:

1. Control/status word to hold Tx MB inactive (CODE = 1000)
2. ID word
3. Data bytes
4. Control/status word (active CODE, LENGTH)

NOTE

The first and last steps are mandatory.

The first write to the control/status word is important in case there was pending reception or transmission. The write operation immediately deactivates the MB, removing it from any currently ongoing arbitration or ID matching processes, giving time for the CPU to program the rest of the MB (see Section 25.6.5.2, “Message Buffer Deactivation”). Once the MB is activated in the fourth step, it will participate in the arbitration process and eventually be transmitted according to its priority. At the end of the successful transmission, the value of the free running timer (TIMER*n*) is written into the message buffer’s time stamp field, the code field in the control and status word is updated, a status flag is set in the IFLAG*n* register, and an interrupt is generated if allowed by the corresponding IMASK*n* register bit. The new code field after transmission depends on the code that was used to activate the MB in step four (see Table 25-13).

25.6.2 Arbitration Process

The arbitration process is an algorithm executed by the message buffer management (MBM) that scans the entire MB memory looking for the highest priority message to be transmitted. All MBs programmed as transmit buffers will be scanned to find the lowest ID or the lowest MB number, depending on the CANCTRL*n*[LBUF] bit.

NOTE

If CANCTRL*n*[LBUF] is cleared, the arbitration considers not only the ID, but also the RTR and IDE bits placed inside the ID at the same positions they are transmitted in the CAN frame.

The arbitration process is triggered in the following events:

- During the CRC field of the CAN frame
- During the error delimiter field of the CAN frame
- During intermission, if the winner MB defined in a previous arbitration was deactivated, or if there was no MB to transmit, but the CPU wrote to the C/S word of any MB after the previous arbitration finished
- When MBM is in idle or bus off state and the CPU writes to the C/S word of any MB
- Upon leaving freeze mode

Once the highest priority MB is selected, it is transferred to a temporary storage space called serial message buffer (SMB), which has the same structure as a normal MB but is not user accessible. This operation is called ‘move-out.’ At the first opportunity window on the CAN bus, the message on the SMB is transmitted according to the CAN protocol rules. FlexCAN transmits up to 8 data bytes, even if the data length code (DLC) value is bigger. Refer to Section 25.6.5.1, “Serial Message Buffers (SMBs),” for more information on serial message buffers.

25.6.3 Receive Process

The CPU prepares or changes an MB for frame reception by writing the following:

1. Control/status word to hold Rx MB inactive (CODE = 0000)
2. ID word
3. Control/status word to mark the Rx MB as active and empty (CODE = 1000)

NOTE

The first and last steps are mandatory.

The first write to the control/status word is important in case there was a pending reception or transmission. The write operation immediately deactivates the MB, removing it from any currently ongoing arbitration or matching process, giving time for the CPU to program the rest of the MB. Once the MB is activated in the third step, it will be able to receive CAN frames that match the programmed ID. At the end of a successful reception, the value of the free running timer (TIMER*n*) is written into the time stamp field, the received ID, data (8 bytes at most) and length fields are stored, the CODE field in the control and status word is updated (see Table 25-12), and a status flag is set in the IFLAG*n* register and an interrupt is generated if allowed by the corresponding IMASK*n* bit.

The CPU should read a receive frame from its MB by reading the following:

1. Control/status word (mandatory—activates internal lock for this buffer)
2. ID (optional—needed only if a mask was used)
3. Data field words
4. Free-running timer (Releases internal lock—optional)

Upon reading the control and status word, if the BUSY bit is set in the CODE field, then the CPU should defer the access to the MB until this bit is negated. Reading the free running timer is not mandatory. If not executed the MB remains locked, unless the CPU reads the C/S word of another MB. Note that only a single MB is locked at a time. The only mandatory CPU read operation is the one on the control and status word to assure data coherency.

The CPU should synchronize to frame reception by an IFLAG*n* bit for the specific MB (see Section 25.5.8, “Interrupt Flag Register (IFLAGn)”), and not by the control/status word CODE field for that MB. Polling the CODE field does not work because once a frame was received and the CPU services the MB (by reading the C/S word followed by unlocking the MB), the CODE field will not return to EMPTY. It will remain FULL, as explained in Table 25-12. If the CPU tries to workaround this behavior by writing to the C/S word to force an EMPTY code after reading the MB, the MB is actually deactivated from any currently ongoing matching process. As a result, a newly received frame matching the ID of that MB may be lost. In summary, never do polling by directly reading the C/S word of the MBs. Instead, read the IFLAG*n* register.

Note that the received identifier field is always stored in the matching MB, thus the contents of the ID field in an MB may change if the match was due to masking.

25.6.3.1 Self-Received Frames

Self-received frames are frames that are sent by the FlexCAN and received by itself. The FlexCAN sends a frame externally through the physical layer onto the CAN bus, and if the ID of the frame matches the ID of the FlexCAN MB, then the frame will be received by the FlexCAN. Such a frame is a self-received frame. Note that FlexCAN does not receive frames transmitted by itself if another device on the CAN bus has an ID that matches the FlexCAN Rx MB ID.

25.6.4 Matching Process

The matching process is an algorithm that scans the entire MB memory looking for Rx MBs programmed with the same ID as the one received from the CAN bus. Only MBs programmed to receive will participate in the matching process for received frames.

While the ID, DLC and data fields are retrieved from the CAN bus, they are stored temporarily in the serial message buffer (Section 25.6.5.1, “Serial Message Buffers (SMBs)”). The matching process takes place during the CRC field. If a matching ID is found in one of the MBs, the contents of the SMB will be transferred to the matched MB during the sixth bit of the end-of-frame field of the CAN protocol. This operation is called ‘move-in.’ If any protocol error (CRC, ACK, etc.) is detected, than the move-in operation does not happen.

An MB with a matching ID is free to receive a new frame if the MB is not locked (see Section 25.6.5.3, “Locking and Releasing Message Buffers”). The CODE field is either EMPTY, FULL, or OVERRUN but the CPU has already serviced the MB (read the C/S word and then unlocked the MB).

Matching to a range of IDs is possible by using ID acceptance masks (RXGMASK*n*, RX14MASK*n*, and RX15MASK*n*). During the matching algorithm, if a mask bit is asserted, then the corresponding ID bit is compared. If the mask bit is negated, the corresponding ID bit is ‘don’t care.’

25.6.5 Message Buffer Handling

In order to maintain data coherency and FlexCAN proper operation, the CPU must obey the rules described in Section 25.6.1, “Transmit Process” and Section 25.6.3, “Receive Process.” Any form of CPU accessing a MB structure within FlexCAN other than those specified may cause FlexCAN to behave in an unpredictable way.

25.6.5.1 Serial Message Buffers (SMBs)

To allow double buffering of messages, the FlexCAN has two shadow buffers called serial message buffers. These two buffers are used by the FlexCAN for buffering both received messages and messages to be transmitted. Only one SMB is active at a time, and its function depends upon the operation of the FlexCAN at that time. At no time does the user have access to or visibility of these two buffers.

25.6.5.2 Message Buffer Deactivation

If the CPU wants to change the function of an active MB, the recommended procedure is to put the module into freeze mode and then change the CODE field of that MB. This is a safe procedure because the FlexCAN waits for pending CAN bus and MB moving activities to finish before entering freeze mode. Nevertheless, a mechanism is provided to maintain data coherence when the CPU writes to the control and status word of active MBs out of freeze mode.

Any CPU write access to the C/S word of an MB causes that MB to be excluded from the transmit or receive processes during the current matching or arbitration round. This mechanism is called MB deactivation. It is temporary, affecting only for the current match/arbitration round.

The purpose of deactivation is data coherency. The match/arbitration process scans the MBs to decide which MB to transmit or receive. If the CPU updates the MB in the middle of a match or arbitration process, the data of that MB may no longer be coherent; therefore, that MB is deactivated.

Even with the coherence mechanism described above, writing to the C/S word of active MBs when not in freeze mode may produce undesirable results. Examples are:

- Matching and arbitration are one-pass processes. If MBs are deactivated after they are scanned, no re-evaluation is done to determine a new match/winner. If an Rx MB with a matching ID is deactivated during the matching process after it was scanned, then this MB is marked as invalid to receive the frame, and FlexCAN will keep looking for another matching MB within the ones it has not scanned yet. If it can not find one, then the message will be lost. Suppose, for example, that two MBs have a matching ID to a received frame, and the user deactivated the first matching MB after FlexCAN has scanned the second. The received frame will be lost even if the second matching MB was free to receive.
- If a Tx MB containing the lowest ID is deactivated after the FlexCAN has scanned it, then the FlexCAN will look for another winner within the MBs that it has not yet scanned. Therefore, it may transmit an MB that may not have the lowest ID at the time because a lower ID might be present that it had already scanned before the deactivation.
- There is a point in time until which the deactivation of a Tx MB causes it not to be transmitted (end of move-out). After this point, it is transmitted, but no interrupt is issued and the CODE field is not updated.

25.6.5.3 Locking and Releasing Message Buffers

Besides message buffer deactivation, the lock/release/busy mechanism is designed to guarantee data coherency during the receive process. The following examples demonstrate how the lock/release/busy mechanism will affect FlexCAN operation:

1. Reading a control/status word of a message buffer triggers a lock for that message buffer. A new received message frame that matches the message buffer cannot be written into this message buffer while it is locked.
2. To release a locked message buffer, the CPU either locks another message buffer (by reading its control/status word) or globally releases any locked message buffer (by reading the free-running timer).
3. If a receive frame with a matching ID is received during the time the message buffer is locked, the receive frame will not be immediately transferred into that message buffer, but will remain in the SMB. There is no indication when this occurs.
4. When a locked message buffer is released, if a frame with a matching identifier exists within the SMB, then this frame will be transferred to the matching message buffer.
5. If two or more receive frames with matching IDs are received while a message buffer with a matching ID is locked, the last received frame with that ID is kept within the serial message buffer, while all preceding ones are lost. There is no indication of lost messages when this occurs.

6. If the user reads the control/status word of a receive message buffer while a frame is being transferred from a serial message buffer, the BUSY code will be indicated. The user should wait until this code is cleared before continuing to read from the message buffer to ensure data coherency. In this situation, the read of the control/status word will not lock the message buffer.

Polling the control/status word of a receive message buffer can lock it, preventing a message from being transferred into that buffer. If the control/status word of a receive message buffer is read, it should then be followed by a read of the control/status word of another buffer, or by reading the free-running timer, to ensure that the locked buffer is unlocked.

NOTE

Deactivation takes precedence over locking. If the CPU deactivates a locked Rx MB, then its lock status is negated, and the MB is marked as invalid for the current matching round. Any pending message on the SMB will not be transferred to the MB anymore.

25.6.6 CAN Protocol Related Frames

25.6.6.1 Remote Frames

The remote frame is a message frame that is transmitted to request a data frame. The FlexCAN can be configured to transmit a data frame automatically in response to a remote frame, or to transmit a remote frame and then wait for the responding data frame to be received.

When transmitting a remote frame, the user initializes a message buffer as a transmit message buffer with the RTR bit set. Once this remote frame is transmitted successfully, the transmit message buffer automatically becomes a receive message buffer, with the same ID as the remote frame that was transmitted.

When a remote frame is received by the FlexCAN, the remote frame ID is compared to the IDs of all transmit message buffers programmed with a CODE of 1010. If there is an exact matching ID, the data frame in that message buffer is transmitted. If the RTR bit in the matching transmit message buffer is set, the FlexCAN will transmit a remote frame as a response.

A received remote frame is not stored in a receive message buffer. It is only used to trigger the automatic transmission of a frame in response. The mask registers are not used in remote frame ID matching. All ID bits (except RTR) of the incoming received frame must match for the remote frame to trigger a response transmission. The matching message buffer immediately enters the internal arbitration process, but is considered as a normal Tx MB, with no higher priority. The data length of this frame is independent of the data length code (DLC) field in the remote frame that initiated its transmission.

25.6.6.2 Overload Frames

Overload frame transmissions are not initiated by the FlexCAN unless certain conditions are detected on the CAN bus. These conditions include detection of a dominant bit in the following:

- First or second bit of intermission
- Seventh (last) bit of the end-of-frame (EOF) field in receive frames

- Eighth (last) bit of the error frame delimiter or overload frame delimiter

25.6.7 Time Stamp

The value of TIMER*n* is sampled at the beginning of the identifier field on the CAN bus. For a message being received, the time stamp will be stored in the TIMESTAMP entry of the receive message buffer at the time the message is written into that buffer. For a message being transmitted, the TIMESTAMP entry will be written into the transmit message buffer once the transmission has completed successfully.

The free-running timer can optionally be reset upon the reception of a frame into message buffer 0. This feature allows network time synchronization to be performed. See the CANCTRL*n*[TSYN] bit.

25.6.8 Bit Timing

The FlexCAN module CANCTRL*n* register configures the bit timing parameters required by the CAN protocol. The CLK_SRC, PRESDIV, RJW, PSEG1, PSEG2, and the PROPSEG fields allow the user to configure the bit timing parameters.

The CANCTRL*n*[CLK_SRC] bit defines whether the module uses the internal bus clock or the output of the crystal oscillator via the CRIN pin. The crystal oscillator clock should be selected whenever a tight tolerance (up to 0.1%) is required for the CAN bus timing. The crystal oscillator clock has better jitter performance than PLL generated clocks. The value of this bit should not be changed, unless the module is in disable mode (CANMCR*n*[MDIS] bit is set)

Careful consideration should be given when selecting the external clocking scheme, the audio sections may require specific input frequencies—for example, 11.2896MHz, that are not suited to meeting the CAN timing requirements, particularly at high speed communication, therefore it may required to use the external audio clock pin when using the CAN modules.

The PRESDIV field controls a prescaler that generates the serial clock (S-clock), whose period defines the time quantum used to compose the CAN waveform. A time quantum is the atomic unit of time handled by the CAN engine.

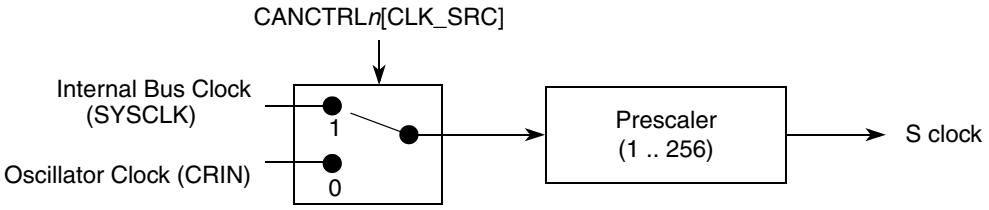


Figure 25-14. CAN Engine Clocking Scheme

$$f_{Tq} = \frac{\text{SYCLK or EXTAL}}{(\text{PRESDIV} + 1)} \qquad \textit{Eqn. 25-5}$$

A bit time is subdivided into three segments[1] (reference Figure 25-15 and Table 25-14):

- SYNC_SEG: Has a fixed length of one time quantum. Signal edges are expected to happen within this section.

1. For further explanation of the underlying concepts please refer to ISO/DIS 11519–1, Section 10.3. Reference also the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

- Time Segment 1: Includes the propagation segment and the phase segment 1 of the CAN standard. It can be programmed by setting the PROPSEG and the PSEG1 fields of the CANCTRL*n* register so that their sum (plus 2) is in the range of 4 to 16 time quanta.
- Time Segment 2: Represents the phase segment 2 of the CAN standard. It can be programmed by setting the PSEG2 field of the CANCTRL*n* register (plus 1) to be 2 to 8 time quanta long.

$$\text{Bit Rate} = \frac{f_{Tq}}{(\text{number of Time Quanta})} \qquad \textit{Eqn. 25-6}$$

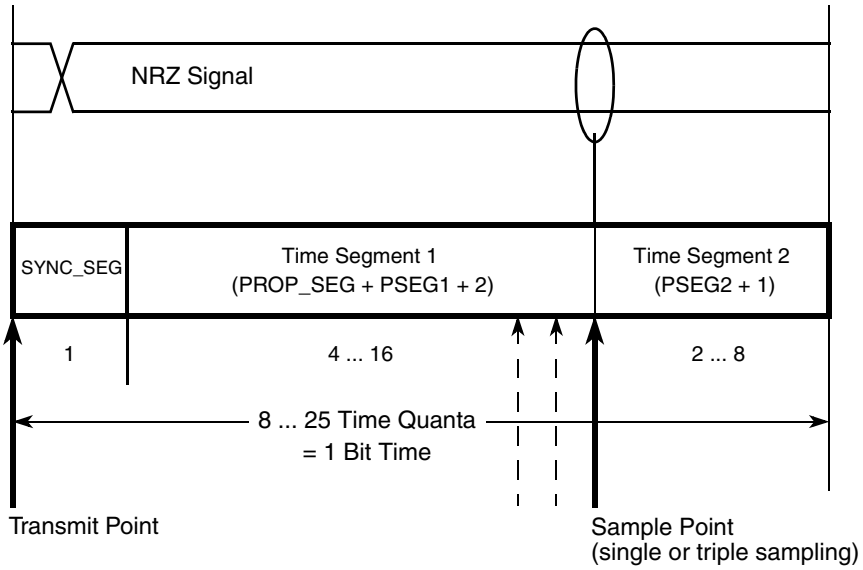


Figure 25-15. Segments within the Bit Time

Table 25-14. Time Segment Syntax

Syntax	Description
SYNC_SEG	System expects transitions to occur on the bus during this period.
Transmit Point	A node in transmit mode transfers a new value to the CAN bus at this point.
Sample Point	A node samples the bus at this point. If the three samples per bit option is selected, then this point marks the position of the third sample.

Table 25-15 gives an overview of the CAN compliant segment settings and the related parameter values.

NOTE

It is the user's responsibility to ensure the bit time settings are in compliance with the CAN standard. For bit time calculations, use an IPT (Information Processing Time) of 2, which is the value implemented in the FlexCAN module

Table 25-15. CAN Standard Compliant Bit Time Segment Settings

Time Segment 1	Time Segment 2	Re-synchronization Jump Width
5 .. 10	2	1 .. 2
4 .. 11	3	1 .. 3
5 .. 12	4	1 .. 4
6 .. 13	5	1 .. 4
7 .. 14	6	1 .. 4
8 .. 15	7	1 .. 4
9 .. 16	8	1 .. 4

25.7 FlexCAN Initialization Sequence

Initialization of the FlexCAN includes the initial configuration of the message buffers and configuration of the CAN communication parameters following a reset, as well as any reconfiguration that may be required during operation. The FlexCAN module may be reset in three ways:

- Device level hard reset—resets all memory mapped registers asynchronously
- Device level soft reset—resets some of the memory mapped registers synchronously (refer to Table 25-1 to see which registers are affected by soft reset)
- CANMCR*n*[SOFT_RST] bit—has the same effect as the device level soft reset

Soft reset is synchronous and has to follow an internal request/acknowledge procedure across clock domains. Therefore, it may take some time to fully propagate its effects. The CANMCR*n*[SOFT_RST] bit remains asserted while soft reset is pending, so software can poll this bit to know when the reset has completed. Also, soft reset can not be applied while clocks are shut down in any of the low power modes. The low power mode should be exited and the clocks resumed before applying soft reset.

The clock source, CANCTRL*n*[CLK_SRC], should be selected while the module is in disable mode. After the clock source is selected and the module is enabled (CANMCR*n*[MDIS] bit cleared), the FlexCAN automatically enters freeze mode. In freeze mode, the FlexCAN is un-synchronized to the CAN bus, the CANMCR*n* register's HALT and FRZ bits are set, the internal state machines are disabled, and the CANMCR*n* register's FRZ_ACK and NOT_RDY bits are set. The CAN*n*TX pin is in recessive state and the FlexCAN does not initiate any transmission or reception of CAN frames. Note that the message buffers are not affected by reset, so they are not automatically initialized.

For any configuration change/initialization, the FlexCAN must be in freeze mode (see Section 25.3.2.2, "Freeze Mode"). The following is a generic initialization sequence applicable to the FlexCAN module:

1. Initialize all operation modes in the CANCTRL*n* register.
   a) Initialize the bit timing parameters PROPSEG, PSEGS1, PSEG2, and RJW.
   b) Select the S-clock rate by programming the PRESDIV field.
   c) Select the internal arbitration mode via the LBUF bit.

2. Initialize message buffers.
   a) The control/status word of all message buffers must be written as either an active or inactive message buffer.
   b) All other entries in each message buffer should be initialized as required.
3. Initialize RXGMASK*n*, RX14MASK*n*, and RX15MASK*n* registers for acceptance mask as needed.
4. Initialize FlexCAN interrupt handler.
   a) Initialize the interrupt controller registers for any needed interrupts. See Chapter 9, "System Integration Module (SIM)," for more information.
   b) Set the required mask bits in the IMASK*n* register (for all message buffer interrupts) and the CANCTRL*n* (for bus off and error interrupts).
5. Clear the CANMCR*n*[HALT] bit. At this point, the FlexCAN will attempt to synchronize with the CAN bus.

25.7.1 Interrupts

There are two interrupt sources for the FlexCAN module. A combined interrupt for all 32 MBs is generated by combining all the interrupt sources from MBs. This interrupt gets generated when any of the 32 MB interrupt sources generates a interrupt. In this case, the CPU must read the IFLAG*n* register to determine which MB caused the interrupt. The other interrupt source (wired OR of bus off and error) acts in the same way, located in the ERRSTAT*n* register. The bus off and error interrupt mask bits are located in the CANCTRL*n* register.

Chapter 26
Real-Time Clock

This chapter provides the external signal descriptions, memory map, register descriptions, and functional descriptions of the Real-Time Clock module of the MCF5251.

The real time clock (RTC) is a mixed-signal circuit which provides an indicator of time (in seconds) for various purposes in the system. The RTC does not stop running, even when the rest of the system is powered down, so long as battery voltage (RTC_VDDA) remains above a certain threshold.

26.1 Block Diagram

Figure 26-1 is a block diagram of the functional organization of the RTC block.

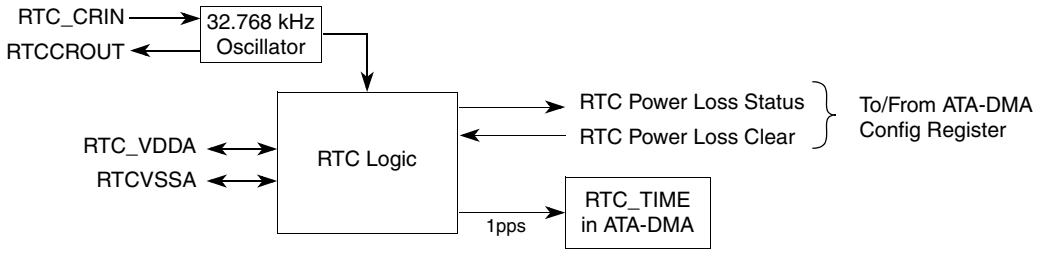


Figure 26-1. Real Time Clock Block Diagram

26.2 External Signal Description

Table 26-1 describes the RTC external signals.

Table 26-1. RTC Signals

Signal Name	Abbreviation	Function	I/O
RTC Crystal In	RTC_CRIN	32.768 kHz crystal input clock.	I
RTC Crystal Out	RTCCROUT	32.768 kHz crystal output.	O
RTC Supply Voltage	RTC_VDDA	Battery supply to RTC	–
RTC Ground	RTCVSSA	Ground for RTC supply	–

26.3 Memory Map and Register Definitions

The RTC module's register is accessible via the USB, ATA DMA, and clock integration module memory space. An RTC control and an RTC status bit are also included in the miscellaneous configuration register.

Table 26-2. Real Time Clock Memory Map

MBAR2 Offset	Register	Access	Reset Value	Section/Page
0x500	Miscellaneous Configuration Register	R/W	Undefined	26.3.1/26-2
0x50C	RTC Time Register (RTC_TIME)	R	Undefined	26.3.2/26-2

26.3.1 Miscellaneous Configuration Register (MISCCR)

See Chapter 22, "USB, ATA DMA, and Clock Integration Module" for a detailed description of this register.

26.3.2 RTC Time Register (RTC_TIME)

The RTC_TIME register is used to read the amount of seconds that have passed since battery voltage has been applied to the device. This register cannot be reset.

Offset MBAR2 0x50C (RTC_TIME) 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	RTC_TIME																															
W																																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 26-2. RTC Time (RTC_TIME) Register

Table 26-3. RTC Time (RTC_TIME) Register Field Descriptions

Field	Description
31–0 RTC_TIME	Indicates the time in seconds since battery voltage was first applied to RTC_VDDA. This register can represent up to 2^{32} seconds (~136 years). It is not possible to reset this value. The user must keep a real-time clock offset in non-volatile memory and apply the offset to this value to convert RTC_TIME into a readable format.

26.4 Functional Description

The real-time clock requires an external 32.768 kHz crystal to operate. From this crystal (via the RTCCRIN and RTCCROUT pins), it will produce a free-running 32-bit counter, that increments with a 1-second tick.

26.4.1 Battery Removal Detection

When the real-time clock battery (RTC_VDD) goes low, the real-time clock may stop, and the RTC_TIME value may become corrupt. To indicate the real-time clock has been in low battery state, it will set the RTC_POWER_LOSS status bit in the ATA-DMA configuration register. This bit will remain set, until the user sets the RTC_CLEAR bit in the ATA-DMA configuration register.