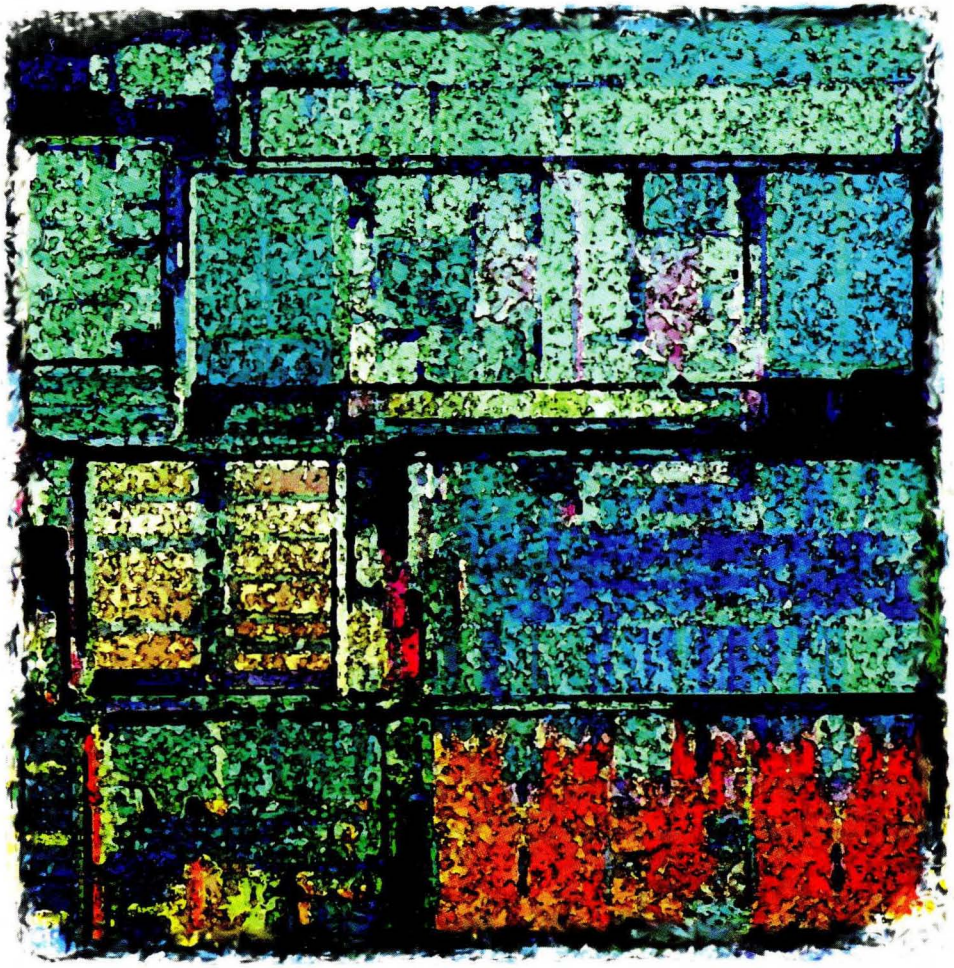
MPC823UM/D

PowerPC™ MPC823

User's Manual



The Microprocessor for Mobile Computing

PowerPC™

MOTOROLA

MOTOROLA

PowerPC™ MPC823 User's Manual

The Microprocessor for Mobile Computing

PREFACE

The *MPC823 User's Manual* discusses the operation, possible configurations, and specifications of the MPC823 microprocessor. This manual mainly addresses Revision A of the silicon, but exceptions are noted where appropriate. All the sections of the manual are organized consistently to make it easier for you to use the manual as a reference. Due to its length, the communication processor module section contains special tabs, links, and page number references that you can use to get to a specific module.

Our website (www.mot.com/mpc823) contains a PDF (3.0) version of the manual that contains hypertext linked cross-references and index markers, which make it easy for you to get in, get the information you need, and get out. However, the links are only operable if you download the file of the entire manual (mpc823.pdf). In the future, we plan to have an online version of this manual in HTML format.

STYLES USED IN THIS MANUAL

Styles make it easier for you to look up the information you need, when you need it. The following components are stylized in this manual:

- Registers are defined in the module to which they belong. If they are mentioned elsewhere, they are referred to by their acronym. It is important to note that register tables include symbols to denote don't care, unaffected, and reserved bits.
  - # indicates that a bit is unaffected
  - — denotes an undefined quality of a bit
  - x indicates that a bit is a "don't care" bit
  - 0 or — indicates that a bit is reserved
- Commands appear in **BOLD UPPERCASE** format.
- Signals, pins, and bit names appear in UPPERCASE format.
- Instructions appear in **bold lowercase** format.

COMPLEMENTARY DOCUMENTATION

Throughout this manual, there are references to the following documents that would enable you to better understand the MPC823 microprocessor. We recommend that you use these manuals in conjunction with the *MPC823 User's Manual*.

- *PowerPC™ Microprocessor Family: The Programming Environments for 32-Bit Microprocessors* (available from your local Motorola sales office using part number MPCFPE32B/AD as a reference)
- USB Specification (available from www.usb.org)

ELECTRICAL SPECIFICATIONS

The most up-to-date electrical specifications for the MPC823 is located on our website.

CUSTOMER FEEDBACK

Motorola welcomes any feedback from our customers. To continuously improve our documentation, we need to know what our customers need and want. So visit our website and tell us!

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

Section 12
System Interface Unit

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

Section 16
Communication Processor Module

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

Section 17
PCMCIA Interface

TABLE OF CONTENTS (Continued)

Section 18
LCD Controller

TABLE OF CONTENTS (Continued)

Section 19
Video Controller

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF TABLES (Continued)

LIST OF TABLES (Continued)

SECTION 1
INTRODUCTION

The MPC823 microprocessor is a versatile, one-chip integrated microprocessor and peripheral combination that can be used in a variety of portable electronic products. It is a low-cost version of the MPC821 microprocessor, except it has been enhanced with additional communication and display capabilities. Specifically, it supports the universal serial bus and video display systems and the existing LCD interface on the MPC821 device. The MPC823 microprocessor particularly excels in low-power, portable, image capture, and personal communication products. It integrates a high-performance embedded PowerPC™ core with a communication processor module that uses a specialized RISC processor for imaging and communication. The communication processor module can perform embedded signal processing functions for image compression and decompression and supports six serial channels—one serial communication controller, two serial management controllers, one I^2C port, one universal serial bus channel, and one serial peripheral interface. This two-processor architecture consumes power more efficiently than traditional architectures because the communication processor module frees the core from peripheral responsibilities like imaging and communication.

1.1 FEATURES

The following list summarizes the main features of the MPC823:

- Embedded PowerPC Core Provides 66MIPS (Using Dhrystone 2.1) or 115K Dhrystones 2.1 at 50MHz
  - ❑ Single-Issue, 32-Bit Version of the PowerPC Core (Fully Compatible with the PowerPC Architecture Definition) with 32 x 32-Bit Fixed-Point Registers
  - ❑ Low Power Consumption, 2.2V Internal, 3.3V I/O Boundary with Microprocessor Core, Caches, Memory Management, and I/O in Operation
  - ❑ Performs Branch Folding, Branch Prediction with Conditional Prefetch, without Conditional Execution
  - ❑ 1K Data Cache and 2K Instruction Cache
  - ❑ Instruction and Data Caches are Two-Way, Set-Associative, Physical Address, 4-Word Line Burst, LRU Replacement Algorithm, Lockable Online Granularity
  - ❑ Memory Management Units with 8-Entry Translation Lookaside Buffers (TLBs) and Fully Associative Instruction and Data TLBs
  - ❑ Memory Management Units Support Multiple Page Sizes of 4K, 16K, 512K and 8M (1K Protection Granularity at the 4K Page Size); 16 Virtual Address Spaces and 16 Protection Groups

- Advanced On-Chip Emulation Debug Mode
- Data Bus Dynamic Bus Sizing for 8-,16-, and 32-Bit Buses
  - ❑ Supports Traditional 68K Big-Endian, Traditional x86 Little-Endian, and PowerPC Little-Endian Memory Systems
  - ❑ Twenty-Six External Address Lines
- Completely Static Design (0–50MHz Operation)
- Communication Processor Module
  - ❑ Interfaces to PowerPC Core Through On-Chip Dual-Access RAM and Virtual (Serial) DMA Channels on a Dedicated DMA Accelerator
  - ❑ Programmable Memory-to-Memory and Memory-to-I/O (Including Flyby) DMA Provided by Virtual DMA Support
  - ❑ 50MIPS @ 15MHz
  - ❑ Protocols Supported by ROM or Download Microcode and the Single Hardware Serial Communication Controller Include, But Are Not Limited To, the Digital Portions of:
    - — Ethernet/IEEE 802.3 (CS/CDMA)
    - — HDLC/SDLC and HDLC Bus
    - — Appletalk
    - — Universal Asynchronous Receiver Transmitter (UART)
    - — Synchronous UART (USART)
    - — Totally Transparent Mode With/Without CRC
    - — Asynchronous HDLC
    - — IrDA Version 1.1 Serial Infrared
    - — Basic Rate ISDN (BRI) in Conjunction with Serial Management Controller Channels
    - — Primary Rate ISDN
  - ❑ 16 x 16-Bit Multiply Accumulate (MAC) Hardware
    - — One Operation Per Clock
    - — Two Clock Latency and One Clock Blockage
    - — Operates Concurrently with Other Instructions
    - — Uses DMA Controller to Burst Data Directly into Register File without Interacting with the PowerPC Core
  - ❑ DSP Functions Are Supported by ROM or Download Microcode and the Communication Processor Module DSP Capabilities, Include, But Are Not Limited To:
    - — V.32bis Datapump Filters
    - — V.34bis Datapump Filters
    - — JPEG Compression/Decompression
  - ❑ 8K Dual-Port RAM. Revision 0 Contains 5K Dual-Port RAM.
  - ❑ Twelve Serial DMA (SDMA) Channels
  - ❑ 32-Bit, Harvard Architecture, Scalar RISC Microcontroller

- ❑ Communication-Specific Commands
- ❑ Supports Continuous Mode Transmission and Reception on All Serial Channels
- ❑ Each Serial Channel has Externally Accessible Pins

- Four Baud Rate Generators. If you are using Mask Revision Base #F98S, there are only two baud rate generators.
  - ❑ Independent and Can Be Connected to A Serial Communication Controller or Serial Management Controller
  - ❑ Allows Changes During Operation
  - ❑ Autobaud Support Option

- One Serial Communication Controller
  - ❑ Ethernet/IEEE 802.3 Support (10Mbps and Full-Duplex Operation)
  - ❑ GeoPort Support
  - ❑ HDLC Bus Implements an HDLC-Based Local Area Network
  - ❑ Universal Asynchronous Receiver Transmitter
  - ❑ Synchronous UART
  - ❑ Serial Infrared (IrDA) Supporting a Maximum of 4Mbps
  - ❑ Totally Transparent. Frame Based with Optional Cyclical Redundancy Check
  - ❑ Maximum Serial Data Rate of 22Mbps

- One Dedicated High-Speed Serial Channel for the Universal Serial Bus (USB)
  - ❑ Supports USB Slave Mode At a Maximum of 12Mbps With Four USB Endpoints

- Two Serial Management Controllers with Externally Accessible Pins
  - ❑ UART
  - ❑ Transparent
  - ❑ General Circuit Interface (GCI) Controller
  - ❑ Can Be Connected to the Time-Division Multiplexed (TDM) Channels

- One Serial Peripheral Interface
  - ❑ Supports Master and Slave Modes
  - ❑ Supports Multimaster Operation on the Same Bus

- One I^2C® Port
  - ❑ Supports Master and Slave Modes
  - ❑ Supports Multimaster Environments
  - ❑ Supports High-Speed Operation
  - ❑ Supports 7-Bit Addressing

- Serial Interface with the Time-Slot Assigner
  - ❑ Allows Serial Communication Controller and Serial Management Controllers To Be Used in Multiplexed and/or Nonmultiplexed Operation
  - ❑ Supports T1, CEPT, PCM Highway, ISDN Basic Rate, ISDN Primary Rate, User-Defined
  - ❑ 1- or 8-Bit Resolution
  - ❑ Allows Independent Transmit and Receive Routing, Frame Syncs, and Clocking
  - ❑ Allows Dynamic Changes
  - ❑ Can Be Internally Connected to Three Serial Channels
- General-Purpose Timers
  - ❑ Four 16-Bit Timers or Two 32-Bit Timers. If you are using Mask Revision Base #F98S, there are only two timers.
  - ❑ Gate Mode Can Enable/Disable Counting
  - ❑ Interrupt Can Be Masked on Reference Match and Event Capture
- Interrupts
  - ❑ Seven External Interrupt Request (IRQ) Lines
  - ❑ One Nonmaskable Interrupt
  - ❑ Twelve Port Pins with Interrupt Capability
  - ❑ Ten Internal Interrupt Sources
  - ❑ Programmable Highest Priority Request
- Memory Controller (Eight Banks)
  - ❑ Can be Programmed to Support Almost any Memory Interface
  - ❑ Each Bank Can Be a Chip-Select or $\overline{RAS}$ to Support a DRAM Bank
  - ❑ A Maximum of 30 Wait States per Memory Bank Can Be Programmed
  - ❑ Glueless Interface to DRAM Single In-Line Memory Modules, Static RAM, Electrically Programmable Read-Only Memory, Flash EPROM.
  - ❑ Four $\overline{CAS}$ lines, Four $\overline{WE}$ lines, and One $\overline{OE}$ Line
  - ❑ Boot Chip-Select Available at Reset (Options for 8-, 16-, or 32-Bit Memory)
  - ❑ Variable Block Sizes—32K to 256M
  - ❑ Selectable Write Protection
  - ❑ On-Chip Bus Arbitration Supports External Bus Master
  - ❑ Special Features for Burst Mode Support
- System Integration Unit
  - ❑ Hardware Bus Monitor
  - ❑ Spurious Interrupt Monitor
  - ❑ Software Watchdog Timer
  - ❑ Periodic Interrupt Timer
  - ❑ Low-Power Stop Mode
  - ❑ Clock Synthesizer

- ❑ PowerPC Decrementer
- ❑ PowerPC Timebase
- ❑ Real-Time Clock
- ❑ Reset Controller
- ❑ IEEE 1149.1 Test Access Port (JTAG)

- Video/LCD Controller
  - ❑ Video Controller
    - — Supports Digital NTSC/PAL Video Encoders and Digital TFT
    - — Sequential RGB, 4:4:4, and 4:2:2 YC_rC_b (CCIR 601) Digital Component Video Formats
    - — CCIR-656 Compatible 8-Bit Interface Port
    - — Horizontal Sync, Vertical Sync, Field and Blanking Timing Generation with Half-Clock Resolution and Programmable Polarity
    - — Supports Interlace/Noninterlace Scanning Methods
    - — Programmable Display Active Area
    - — Programmable Background Color for Inactive Area
    - — Glueless Interface for Most Digital Video Encoders
    - — Hardware Horizontal Scrolling
    - — Uses Burst Read DMA Cycles for Maximum Bus Performance
    - — Panel Voltage Control Adjustments for Contrast Set with On-Chip Timers
    - — End-of-Frame Interrupt Generation
  - ❑ LCD Controller
    - — Supports Digital TFT and Passive LCD Panels
    - — Horizontal Sync, Vertical Sync, Field and Blanking Timing Generation with Half-Clock Resolution and Programmable Polarity
    - — 1-, 2-, or 4-Bit Per Pixel Grayscale Mode Using Advanced Frame Rate Control (FRC) Algorithm
    - — Four or Eight Bits Per Pixel Color Mode
    - — 4-, 8-, 9-, or 12-Bit Parallel Output to LCD Displays
    - — Programmable Display Active Area
    - — Non-Split or Vertically-Split Screen Support
    - — Uses Burst Read DMA Cycles for Maximum Bus Performance
    - — End of Frame Interrupt Generation
    - — Data for Splits—2+2 or 4+4 Parallel Bits (x+x Refers to x Bits Each for Lower and Upper Screens in Parallel)
    - — Built-In Color RAM with 256 12-Bit Entries
    - — Programmable Wait Time Between Lines and Frames
    - — Panel Voltage Control Adjustments for Contrast Set with On-Chip Timers
    - — Programmable Polarity for All LCD Interface Signals

- Single-Socket PCMCIA-ATA Interface
  - ❑ Master Interface, Release 2.1 Compliant
  - ❑ Single PCMCIA Socket
  - ❑ Eight Memory or I/O Windows Available
  - ❑ Eight General-Purpose I/O Pins and Two General-Purpose Output-Only Pins are Available When the PCMCIA Controller Is Not in Operation
- Low-Power Support Modes
  - ❑ Normal High–All Units Are Fully Powered at High Clock Frequency
  - ❑ Normal Low–All Units Are Fully Powered at Low Clock Frequency
  - ❑ Doze–Core Functional Units Are Disabled, Except Timebase, Decrementer, PLL, Memory Controller, Real-Time Clock, LCD, and Communication Processor Module
  - ❑ Sleep–All Units Are Disabled, Except Real-Time Clock, Periodic Interrupt Timer, Timebase, and Decrementer. PLL Is Active for Fast Wake-up
  - ❑ Deep Sleep–All Units Are Disabled Including PLL, But Not the Real-Time Clock and Periodic Interrupt Timer, Timebase, and Decrementer
  - ❑ Power-Down—All Units Are Disabled Including PLL, but Not the Real-Time Clock and Periodic Interrupt Timer, Timebase, and Decrementer. Saves More Power Than Other Modes. The State of Certain Registers May Be Preserved,
  - ❑ Separate Power Supply Input to Operate Internal Logic at 2.2V At or Below 25MHz Operation
  - ❑ Can Be Dynamically Shifted Between High Frequency (3.3V Internal) and Low Frequency (2.2V Internal) Operation
- Development Capabilities and Interface
  - ❑ Program Flow Tracking
    - — Instruction Show Cycle
    - — Data Show Cycle
    - — Branching
    - — Exception Traps
  - ❑ Watchpoints and Breakpoints
    - — Four Hardware Breakpoints
    - — Five Watchpoint Sources
  - ❑ Simple Hardware Interface
    - — High-Speed Data Transfer
    - — Internal Status Pins
    - — Freeze Indication
  - ❑ Rich Control Register Set
- 3.3V Operation with 5V TTL Compatibility for the Parallel Port Pins and 3.3V for All Others. For Revision 0, All Pins are 5V TTL Compatible.
- 256-Pin Plastic Ball Grid Array (BGA) Packaging

1.2 ARCHITECTURE

The MPC823 microprocessor uses a dual-processor architecture design approach to provide you with a high-performance, general-purpose RISC integer processor and a special-purpose 32-bit scalar RISC communication processor module. The peripherals are uniquely designed for communication requirements and can provide embedded signal processing functions for communication and user interface enhancements and the I/O support needed for high-speed digital communications. The MPC823 is comprised of three main modules that interface with the 32-bit internal bus:

- The embedded PowerPC core
- The system interface unit
- The communication processor module
- LCD controller

The MPC823 block diagram is illustrated in Figure 1-1.

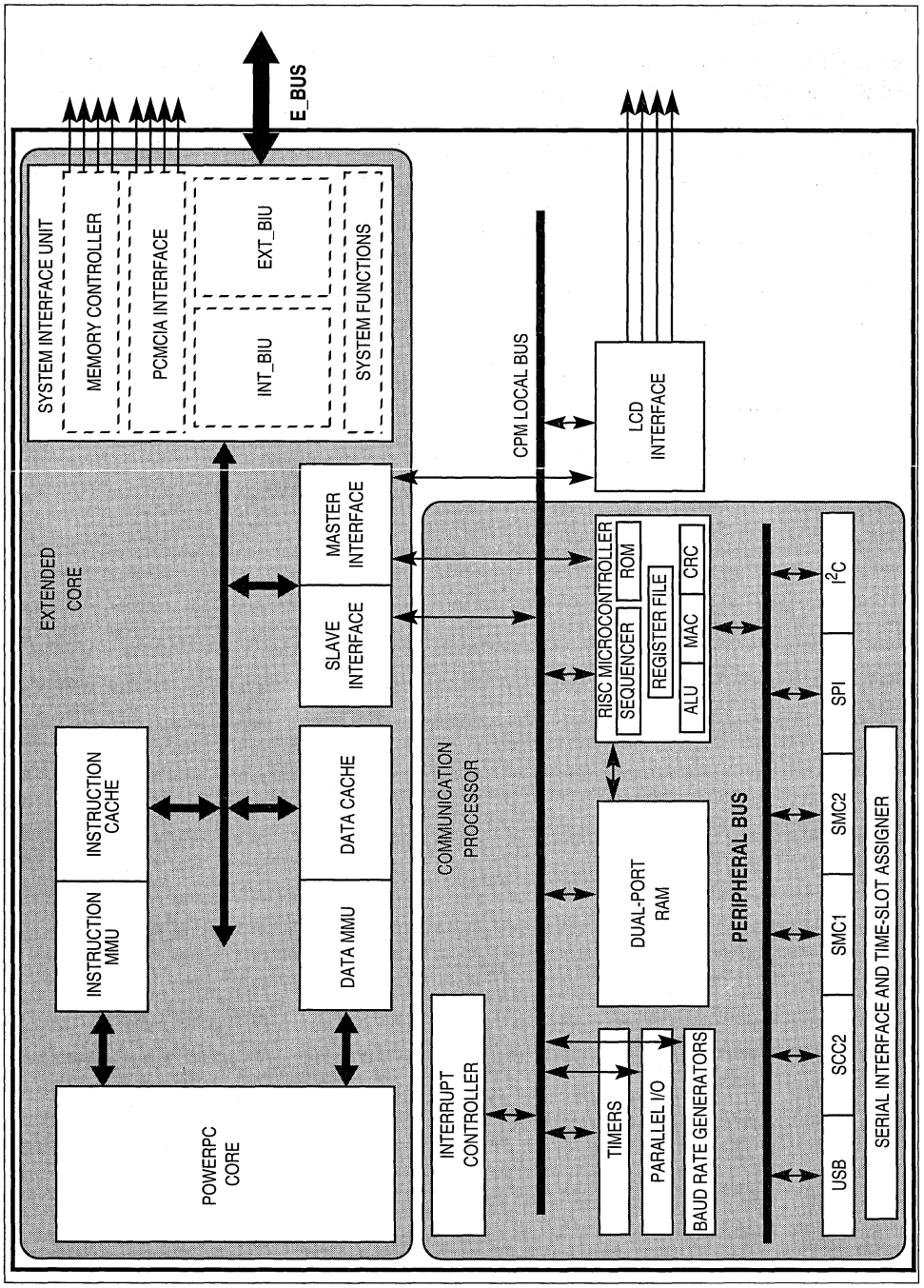


Figure 1-1. MPC823 Block Diagram

1.2.1 The Embedded PowerPC Core

The PowerPC core complies with standard PowerPC architecture. It has a fully static design that consists of three functional blocks—the integer block, hardware multiplier/divider, and load/store block. The core supports integer operations on a 32-bit internal data path and 32-bit arithmetic hardware. Its interface to the internal and external buses is 32 bits. The core uses a two-instruction load/store queue, four-instruction prefetch queue, and a six-instruction history buffer. It performs branch folding and branch prediction with conditional prefetch, but without conditional execution. With single bus cycles, the core can operate on 32-bit external operands and with critical-word-first in multiple bus cycles. The PowerPC integer block supports 32 x 32-bit fixed-point general-purpose registers and can execute one integer instruction per clock cycle.

The PowerPC core is integrated with the memory management units, an instruction cache, and a data cache. The memory management units (MMUs) provide 8-entry, fully-associative instruction and data TLBs, with multiple page sizes of 4K (1K protection), 16K, 512K, and 8M. They support 16 virtual address spaces and 16 protection groups. Special registers are available to support software tablewalk and update.

The instruction cache is 2K, two-way, set-associative with physical addressing. It allows single-cycle accesses on hit with no added latency for miss. It is four words per line and supports burst line fill using an LRU replacement algorithm. The cache can be locked on a line basis for application critical routines. The data cache is 1K, two-way, set-associative with physical addressing. It allows single-cycle accesses on hit with one added clock latency for miss. It has four words per line and supports burst line fill using an LRU replacement algorithm. The cache can be locked on a line basis for application critical data and can be programmed to support copyback or writethrough mode via the memory management unit. The cache-inhibit mode can be programmed per MMU page. The PowerPC core, with its instruction and data caches, can deliver approximately 66 MIPS at 50MHz (using Dhrystone 2.1) or 115K Dhrystones, based on the assumption that it is issuing one instruction per cycle with a cache hit rate of 94%.

1.2.2 The System Interface Unit

The system interface unit supports traditional 68K big-endian memory systems, traditional x86 little-endian memory systems, and PowerPC little-endian memory systems. It also provides power management functions, reset control, a PowerPC decrementer, PowerPC timebase, and real-time clock. Although the PowerPC core is a 32-bit device internally, it can be configured to operate with an 8-, 16-, or 32-bit data bus. Regardless of the system bus size, dynamic bus sizing is supported, which allows 8-, 16-, and 32-bit peripherals and memory to coexist on a 32-bit system bus.

The memory controller supports up to eight memory banks with glueless interfaces to DRAM, SRAM, EPROM, Flash EPROM, SDRAM, EDO and other peripherals with two-clock initial access to external SRAM and bursting support. It provides variable block sizes between 32K and 256M. The memory controller has 0 to 20 wait states for each bank of memory and can use address type matching to qualify each memory bank access. It provides four byte-enable signals for varying width devices, one output-enable signal, and one boot chip-select that is available at reset.

The DRAM interface supports 8-, 16-, and 32-bit ports and uses a programmable state machine to support almost any memory interface. Memory banks can be defined in depths of 256K, 512K, 1M, 2M, 4M, 8M, 16M, 32M, or 64M for all port sizes. In addition, the memory depth can be defined as 64K and 128K for 8-bit memory or 128M and 256M for 32-bit memory. The DRAM controller supports page mode access for successive transfers within bursts. The MPC823 supports a glueless interface to one bank of DRAM, while external buffers are required for additional memory banks. The refresh unit provides $\overline{CAS}$ before $\overline{RAS}$, a programmable refresh timer, refresh active during external reset, disable refresh modes, and stacking for a maximum of seven refresh cycles.

1.2.3 The Communication Processor Module

The communication processor module (CPM) contains features that allow the MPC823 microprocessor to excel in imaging, personal communication, and low-power applications. These features are divided into three categories:

- DSP processing
- Communication processing
- Twelve serial DMA channels and two independent DMA channels

The MPC823's embedded DSP function allows the communication processor module to execute imaging algorithms in parallel with the PowerPC core to achieve maximum performance with very little power. The DSP can execute one 16x16 MAC on every clock cycle. It has preprogrammed filtering functions like FIR, MOD, DEMOD, IIR, and downloadable imaging functions for JPEG image compression and decompression. These functions are also used by modem and speech recognition programs.

The robust communication features of the MPC823 come from the communication processor module. These features include a RISC microcontroller with multiply accumulate (MAC) hardware, one serial communication controller (SCC), two serial management controllers (SMCs), one dedicated serial channel for the universal serial bus (USB), one inter-integrated circuit (I^2C) port, one serial peripheral interface (SPI), 8K dual-port RAM, an interrupt controller, a time-slot assigner, and four independent baud rate generators.

Twelve serial DMA channels support the SCC, SMCs, USB channel, SPI, and I^2C controllers. The independent DMAs give you two channels for general-purpose DMA usage. They offer high-speed transfers, 32-bit data movement, buffer chaining, and independent request and acknowledge logic. The RISC microcontroller is the only block that can access the IDMA registers directly. The CPU can only access them indirectly via a buffer descriptor.

1.2.4 The Video/LCD Controller

The MPC823 has a dual-purpose video/LCD controller that shares common dual-port memory. You can, however, only run one of the controllers at a time.

1.2.4.1 THE VIDEO CONTROLLER. The video controller can be used to drive a digital NTSC/PAL encoder or a wide variety of digital LCD panels. The frame buffer is stored in system memory in the form of an orthogonal matrix—rows and columns. The 24-bit color data is organized as pixel components whether it is sequential RGB or YC_rC_b. Each pixel component is represented by a byte. The video controller uses a dedicated DMA channel to read the display data from the frame buffer and drive it to the video interface. It also generates the required timing signals such as horizontal sync, vertical sync, field, and blanking. Refer to **Section 19 Video Controller** for more information.

1.2.4.2 THE LCD CONTROLLER. The LCD controller provides extremely versatile LCD support for 8-bit color, monochrome or 4/16-level grayscale, color TFT (12 bits, 4x3 RGB), and passive color (xSTN) 4/8 bit data. The controller supports 4-bit single-scan, 8-bit dual-scan, 2+2 bit dual-scan, or 4+4 bit dual-scan. It is programmable for frame rate, number of pixels per line, and number of lines per frame. The panel voltage is programmable through the duty cycle for contrast adjustments implemented in the communication processor module program. Display data is stored in your own memory space and is transferred into the controller using the DMA channel. Refer to **Section 18 LCD Controller** for more information.

1.3 THE PCMCIA-ATA CONTROLLER

The PCMCIA-ATA interface is a master controller that is compliant with Version 2.1. The interface supports one independent PCMCIA socket with the required external transceivers or buffers. It provides eight memory or I/O windows that can be allocated to the socket. If the PCMCIA port is not being used as a card interface, it can provide eight general-purpose pins and two output-only pins with interrupt capability.

1.4 POWER MANAGEMENT

The MPC823 microprocessor supports a wide range of power management features, including normal high, normal low, doze, sleep, deep-sleep, and power-down modes. In normal high mode, the MPC823 microprocessor is fully powered with all internal units operating at the full speed of the processor. Normal low mode is the same as normal high, except it operates at a much lower frequency. There is a doze mode determined by a clock divider that allows the operating system to reduce the operational frequency of the processor.

Doze mode disables core functional units except the timebase, decrementer, PLL, memory controller, real-time clock, LCD controller, and communication processor module. Sleep mode is a lower power mode that disables everything except the real-time clock, timebase, decrementer, and periodic interrupt timer, thus leaving the PLL active for quick wake-up. The deep-sleep mode then disables the PLL for lower power, but slower wake-up. Power-down mode disables all logic in the processor, except the minimum logic required to restart the device. It saves the most power, but requires the longest wake-up time.

The MPC823 microprocessor also provides a separate set of power pins for the internal logic in the device. These power pins can be used to give the device a 2.2V power source that can be used when the microprocessor is operating at 25MHz or less. This capability reduces the power consumption of the device by an additional 30%.

1.5 SYSTEM DEBUG SUPPORT

The MPC823 microprocessor contains an advanced debug interface that provides superior debug capabilities without any loss of speed. It supports six watchpoint pins that can be combined with eight internal comparators, four of which operate on the effective address of the address bus. The other four comparators are split—two comparators operate on the effective address on the data bus and two comparators operate on the data on the data bus. The MPC823 microprocessor can compare using the =, ≠, <, and > conditions to generate watchpoints. Each watchpoint can then generate a breakpoint that can be programmed to trigger in a programmable number of events.

1.6 APPLICATIONS

The MPC823 microprocessor is specifically designed to be a general-purpose, low-cost entry point to the Motorola embedded PowerPC Family for systems in which advanced GUIs, communications, and high-level real-time operating systems are used. The device excels in applications that require the performance of single-issue PowerPC core with a moderate amount of data and instruction cache. It provides all the basic features of glueless memory connections along with highly functional serial connectivity, a graphical LCD, and a video display controller. The MPC823 excels in low-power and portable applications because of its extensive power-down modes and low normal operation current.

1.7 DIFFERENCES BETWEEN THE MPC823 AND MPC821

The MPC823 is mostly a subset of the MPC821 microprocessor. To create the MPC823, the following modifications were made to the MPC821:

- Instruction cache was reduced from 4K to 2K
- Data cache was reduced from 4K to 1K
- Instruction and data MMUs were reduced from 32 to 8 TLB entries each
- Pin count was reduced from 357 pins to 256 pins
- Package size was reduced from 25mm (ZP) to 23mm (ZT)
- Address bus was reduced from 32 bits to 26 bits
- Bisync protocol was removed from the serial communication controller
- PCMCIA support was reduced to a single channel, PCMCIA channel B.
- One serial communication controller was removed from the CPM leaving a single channel (SCC2)
- One USB channel was added to CPM in place of SCC1
- High-speed IRDA support (1.152Mbps and 4Mbps) was added
- Centronics protocol was removed
- Video controller was added to support NTSC/PAL monitors

1.8 MPC823 GLUELESS SYSTEM DESIGN

The MPC823 was primarily designed to make it easy for you to interface a microprocessor with other system components. Figure 1-2 illustrates a system configuration that contains one flash EPROM and yet supports DRAM SIMM and one SRAM. Depending on the capacitance of the system bus, external buffers may be required. From a logic standpoint, however, a glueless system is maintained.

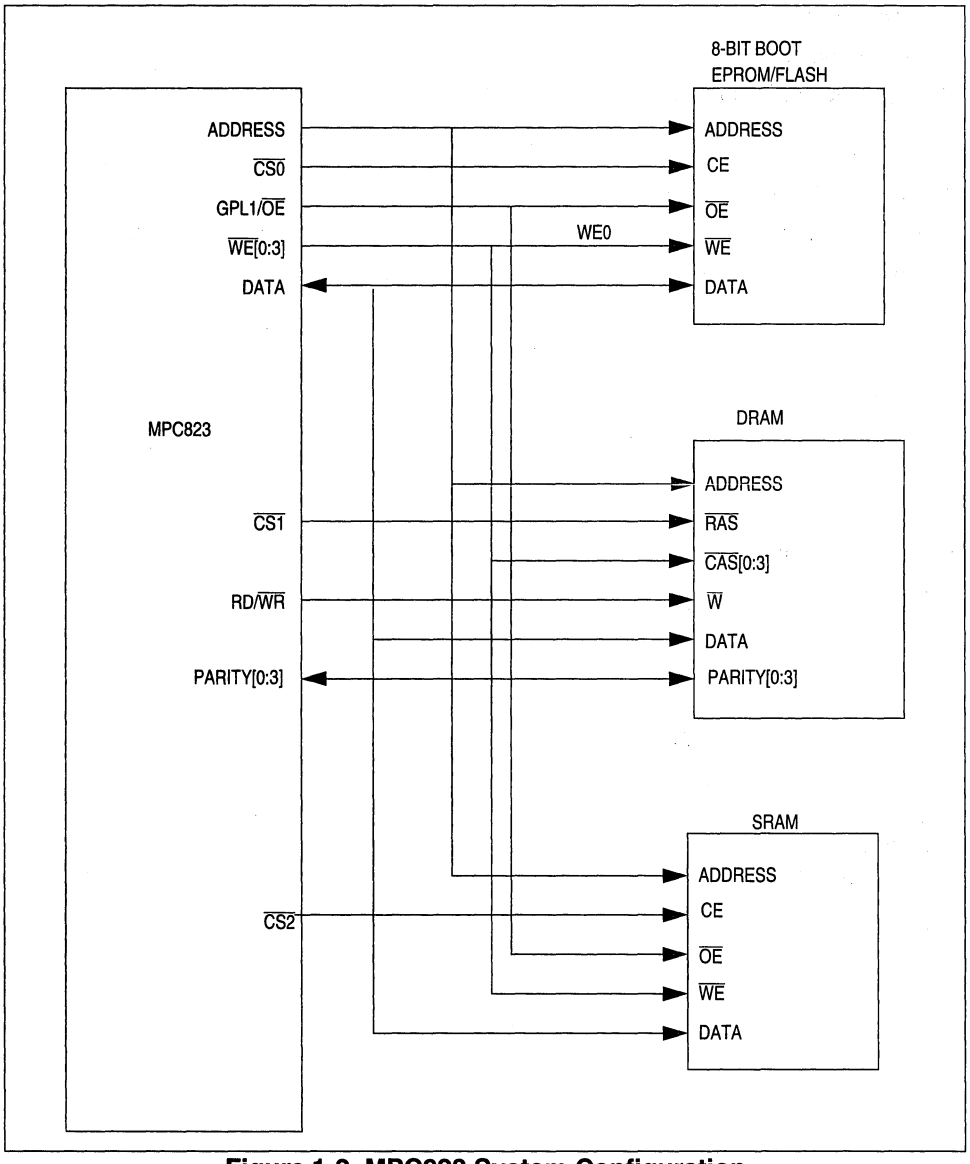


Figure 1-2. MPC823 System Configuration

SECTION 2
EXTERNAL SIGNALS

This section briefly describes each of the MPC823 input and output signals.

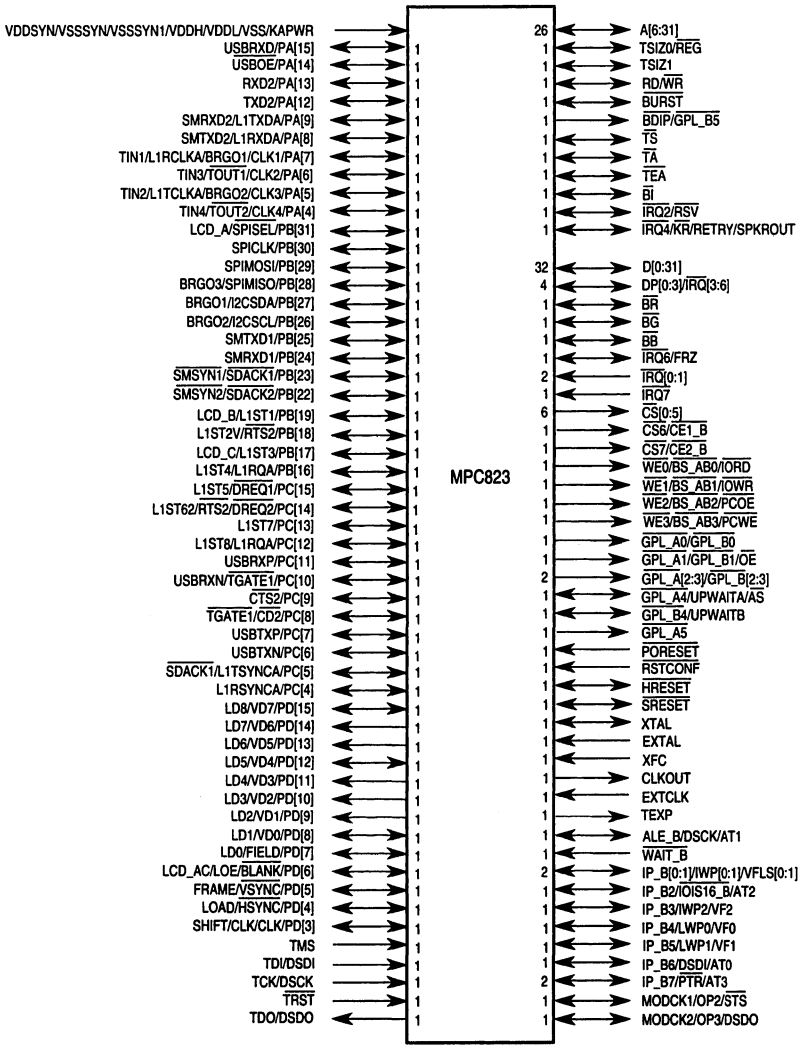


Figure 2-1. MPC823 Signal Pinout

2.1 THE SYSTEM BUS SIGNALS

The MPC823 system bus signals consist of all the lines that interface with the external bus. Many of these lines perform different functions, depending on how you assign them. The following input and output signals are identified by their mnemonic name and each signal's pin number can be found in Figure 2-1.

Table 2-1. Signal Descriptions

SIGNAL	PIN NUMBER	DESCRIPTION
A[6-31]	See Table 2-2 for pin breakout.	**Address Bus**—This bidirectional three-state signal provides the address for the current bus cycle. A0 is the most-significant signal for this bus. The signal is output when an internal master on the MPC823 initiates a transaction on the external bus. The signal is input when an external master initiates a transaction on the bus and it is sampled internally to allow the memory controller/PCMCIA interface to control the accessed slave device.
TSIZ0 $\overline{\text{REG}}$	F15	**Transfer Size 0**—When accessing a slave in the external bus, this three-state signal is used (together with TSIZ1) by the bus master to indicate the number of operand bytes waiting to be transferred in the current bus cycle. This signal is input when an external master initiates a transaction on the bus and it is sampled internally to allow the memory controller/PCMCIA interface to control the accessed slave device. **$\overline{\text{REG}}$**—When the access is initiated by an internal master to a slave under control of the PCMCIA interface, this signal is output to indicate which space in the PCMCIA card is currently accessed.
TSIZ1	E15	**Transfer Size 1**—This three-state signal is used (with TSIZ0) by the bus master to indicate the number of operand bytes waiting to be transferred in the current bus cycle. This signal is driven by the MPC823 when it is the owner of the bus. It is input when an external master initiates a transaction on the bus and it is sampled internally to allow the memory controller/PCMCIA interface to control the accessed slave device.
RD/$\overline{\text{WR}}$	C13	**Read Write**—This three-state signal is driven by the bus master to indicate the direction of the bus's data transfer. A logic one indicates a read from a slave device and a logic zero indicates a write to a slave device. This signal is driven by the MPC823 when it is the owner of the bus. It is input when an external master initiates a transaction on the bus and is sampled internally to allow the memory controller/ PCMCIA interface to control the accessed slave device.
$\overline{\text{BURST}}$	B10	**Burst Transaction**—This three-state signal is driven by the bus master to indicate that the current initiated transfer is a burst one. This signal is driven by the MPC823 when it is the owner of the bus. It is input when an external master initiates a transaction on the bus; this signal and is sampled internally to allow the memory controller/PCMCIA interface to control the accessed slave device.
$\overline{\text{BDIP}}$ $\overline{\text{GPL\_B5}}$	A13	**Burst Data in Progress**—When accessing a slave device in the external bus, the master on the bus asserts this signal to indicate that the data beat in front of the current one is the one requested by the master. This signal is negated prior to the expected last data beat of the burst transfer. **General-Purpose Line B5**—This signal is used by the memory controller when the user programmable machine B (UPMB) takes control of the slave access.
$\overline{\text{TS}}$	D10	**Transfer Start**—This three-state signal is asserted by the bus master to indicate the start of a bus cycle that transfers data to or from a slave device. This signal is driven by the master only when it has gained ownership of the bus. Every master should negate this signal before the bus relinquishes. A pull-up resistor should be connected to this signal to prevent a slave device from detecting a spurious bus accessing it when no master is taking ownership of the bus. This signal is sampled by the MPC823 when it is not the owner of the external bus to allow the memory controller/PCMCIA interface to control the accessed slave device. It indicates that an external synchronous master initiated a transaction.

Table 2-1. Signal Descriptions (Continued)

SIGNAL	PIN NUMBER	DESCRIPTION
$\overline{TA}$	A12	**Transfer Acknowledge**—This bidirectional three-state signal indicates that the slave device addressed in the current transaction has accepted the data transferred by the master (write) or has driven the data bus with valid data (read). The signal behaves as an output when the PCMCIA memory controller takes control of the transaction. The only exception occurs when the memory controller is controlling the slave access by means of the GPCM and the corresponding option register is instructed to wait for an external assertion of the transfer acknowledge line. Every slave device should negate the ta signal after the end of the transaction and immediately three-state it to avoid contentions on the line if a new transfer is initiated addressing other slave devices. A pull-up resistor should be connected to this signal to keep a master device from detecting the assertion of this signal when no slave is addressed in a transfer or when the address detection for the addressed slave is slow.
$\overline{TEA}$	C11	**Transfer Error Acknowledge**—This open-drain signal indicates that a bus error occurred in the current transaction. It is driven asserted by the MPC823 when the bus monitor does not detect a bus cycle termination within a reasonable amount of time. The assertion of $\overline{TEA}$ causes the termination of the current bus cycle, thus ignoring the state of $\overline{TA}$.
$\overline{BI}$	B12	**Burst Inhibit**—This bidirectional three-state signal indicates that the slave device addressed in the current burst transaction is unable to support burst transfers. The signal behaves as an output when the PCMCIA memory controller takes control of the transaction. When the MPC823 drives out the signal for a specific transaction, it asserts or negates $\overline{BI}$ during the transaction according to the value you specify in the appropriate control registers. It negates the signal after the end of the transaction and immediately three-states it to avoid contentions if a new transfer is initiated addressing other slave devices.
$\overline{RSV}$ $\overline{IRQ2}$	D9	**Reservation**—This three-state signal is output by the MPC823 in conjunction with the address bus to indicate that the internal core initiated a transfer as a result of a **stwcx** or **lwarx** instruction. **Interrupt Request 2**—This input is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core.
$\overline{IRQ4}$ $\overline{KR}$ RETRY SPKROUT	B7	**Interrupt Request 4**—This input signal is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core. It should be noted that the interrupt request signal that is sent to the interrupt controller is the logical AND of this signal (if defined to function as $\overline{IRQ4}$) and the DP1/$\overline{IRQ4}$ (if defined to function as $\overline{IRQ4}$). **Kill Reservation**—This input is used as a part of the storage reservation protocol when the MPC823 initiated a transaction as the result of a **stwcx** instruction. **Retry**—This input is used by the slave device to indicate that it is unable to accept the transaction. The MPC823 has to relinquish the ownership of the bus and initiate the transaction again after winning again in the bus arbitration. **Speaker Out**—This output signal is used to provide a digital audio waveform to be driven to the system's speaker.
D[0:31]	See Table 2-2 for pin breakout.	**Data Bus**—This bidirectional three-state signal provides the general-purpose data path between the MPC823 and all other devices. Although the data path is a maximum of 32 bits wide, it can be dynamically sized to support 8-, 16-, or 32-bit transfers. D0 is the most-significant bit of the data bus.
DP0 $\overline{IRQ3}$	C3	**Data Parity 0**—This bidirectional three-state signal provides parity generation and checking for the data bus lane D[0:7] by transferring to a slave device initiated by the MPC823. The parity function can be defined independently for each one of the addressed memory banks (if controlled by the memory controller) and for the rest of the slaves sitting on the external bus. **Interrupt Request 3**—This input signal is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core.
DP1 $\overline{IRQ4}$	D4	**Data Parity 1**—This bidirectional three-state signal provides parity generation and checking for the data bus lane D[8:15] by transferring to a slave device initiated by the MPC823. The parity function can be defined independently for each one of the addressed memory banks (if controlled by the memory controller) and for the rest of the slaves on the external bus. **Interrupt Request 4**—This input is one of the eight external lines that can request (by means of the internal interrupt controller) a service routine from the core. It should be noted that the interrupt request signal that is sent to the interrupt controller is the logical AND of this signal (if defined to function as $\overline{IRQ4}$) and the KR/SPKROUT/ $\overline{IRQ4}$ if defined to function as $\overline{IRQ4}$.

Table 2-1. Signal Descriptions (Continued)

SIGNAL	PIN NUMBER	DESCRIPTION
DP2 IRQ5	D3	**Data Parity 2**—This bidirectional three-state signal provides parity generation and checking for the data bus lane D[16:23] by transferring to a slave device initiated by the MPC823. The parity function can be defined independently for each one of the addressed memory banks (if controlled by the memory controller) and for the rest of the slaves on the external bus. **Interrupt Request 5**—This input signal is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core.
DP3 IRQ6	C2	**Data Parity 3**—This bidirectional three-state signal provides parity generation and checking for the data bus lane D[24:31] by transferring to a slave device initiated by the MPC823. The parity function can be defined independently for each one of the addressed memory banks (if controlled by the memory controller) and for the rest of the slaves on the external bus. **Interrupt Request 6**—This input signal is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core. It should be noted that the interrupt request signal that is sent to the interrupt controller is the logical AND of this signal (if defined to function as IRQ6) and the FRZ/IRQ6 if defined to function as IRQ6.
BR	B11	**Bus Request**—This bidirectional signal is asserted low when a possible master is requesting ownership of the bus. When the MPC823 is configured to operate with the internal arbiter, this signal is configured as an input. However, when the MPC823 is configured to operate with an external arbiter, this signal is configured as an output and asserted every time a new transaction is intended to be initiated and no parking on the bus is granted.
BG	C10	**Bus Grant**—This bidirectional signal is asserted low when the arbiter of the external bus grants the specific master ownership of the bus. When the MPC823 is configured to operate with the internal arbiter, this signal is configured as an output and asserted every time the external master asserts the BR signal and its priority request is higher than any of the internal sources requiring the initiation of a bus transfer. However, when the MPC823 is configured to operate with an external arbiter, this signal is configured as an input.
BB	A11	**Bus Busy**—This bidirectional signal is asserted low by a master to show that it owns the bus. The MPC823 asserts this signal after the bus arbiter grants it bus ownership and the BB signal is negated.
IRQ6 FRZ	A10	**Interrupt Request 6**—This input signal is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core. It should be noted that the interrupt request signal that is sent to the interrupt controller is the logical AND of this signal (if defined to function as IRQ6) and the DP3/IRQ6 (if defined to function as IRQ6.) **Freeze**—This output signal is asserted to indicate that the internal core is in debug mode.
IRQ0	N1	**Interrupt Request 0**—This input signal is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core.
IRQ1	N2	**Interrupt Request 1**—This input signal is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core.
IRQ7	N3	**Interrupt Request 7**—This input signal is one of the eight external signals that can request (by means of the internal interrupt controller) a service routine from the core.
CS[0:5]	See Table 2-2 for pin breakout.	**Chip Select**—These output signals enable peripheral or memory devices at programmed addresses if they are appropriately defined in the memory controller. CS0 can be configured to be the global chip-select for the boot device.
CS6 CE1_B	C14	**Chip Select 6**—This output signal enables a peripheral or memory device at a programmed address if defined appropriately in the BR6 and OR6 of the memory controller. **Card Enable 1 Slot B**—This output signal enables even byte transfers when accesses to the PCMCIA Slot B are handled by the PCMCIA interface.

Table 2-1. Signal Descriptions (Continued)

SIGNAL	PIN NUMBER	DESCRIPTION
$\overline{CS7}$ $\overline{CE2\_B}$	B15	**Chip Select 7**—This output signal enables a peripheral or memory device at a programmed address if defined appropriately in the BR7 and OR7 registers of the memory controller. **Card Enable 2 Slot B**—This output signal enables odd byte transfers when accesses to the PCMCIA Slot B are handled by the PCMCIA interface.
$\overline{WE0}$ $\overline{BS\_AB0}$ $\overline{IORD}$	D16	**Write Enable 0**—This output signal is asserted when a write access to an external slave controlled by the GPCM in the memory controller is initiated by the MPC823. $\overline{WE0}$ is asserted if the data lane D[0:7] contains valid data to be stored by the slave device. **Byte Select 0 on UPMA or UPMB**—This output signal is asserted as required by the UPMA or UPMB in the memory controller whenever you program it. In a read or write transfer, the signal is only asserted if the data lane D[0:7] contains valid data. **I/O Device Read**—This output signal is asserted when the MPC823 initiates a read access to a region controlled by the PCMCIA interface. The signal is only asserted if the access is to a PC Card I/O space.
$\overline{WE1}$ $\overline{BS\_AB1}$ $\overline{IOWR}$	E16	**Write Enable 1**—This output signal is asserted when the MPC823 initiates a write access to an external slave controlled by the GPCM in the memory controller. $\overline{WE1}$ is asserted if the data lane D[8:15] contains valid data to be stored by the slave device. **Byte Select 1 on UPMA or UPMB**—This output signal is asserted as required by the UPMA or UPMB in the memory controller whenever you program it. In a read or write transfer, the signal is only asserted if the data lane D[8:15] contains valid data. **I/O Device Write**—This output signal is asserted when the MPC823 initiates a write access to a region controlled by the PCMCIA interface. The signal is only asserted if the access is to a PC Card I/O space.
$\overline{WE2}$ $\overline{BS\_AB2}$ $\overline{PCOE}$	D15	**Write Enable 2**—This output signal is asserted when the MPC823 initiates a write access to an external slave controlled by the GPCM in the memory controller. $\overline{WE2}$ is asserted if the data lane D[16:23] contains valid data to be stored by the slave device. **Byte Select 2 on UPMA or UPMB**—This output signal is asserted as required by the UPMA or UPMB in the memory controller whenever you program it. In a read or write transfer, the signal is only asserted if the data lane D[16:23] contains valid data. **PCMCIA Output Enable**—This output signal is asserted when the MPC823 initiates a read access to a memory region under the control of the PCMCIA interface.
$\overline{WE3}$ $\overline{BS\_AB3}$ $\overline{PCWE}$	F13	**Write Enable 3**—This output signal is asserted when the MPC823 initiates a write access to an external slave controlled by the GPCM in the memory controller. $\overline{WE3}$ is asserted if the data lane D[24:31] contains valid data to be stored by the slave device. **Byte Select 3 on UPMA or UPMB**—This output signal is asserted as required by the UPMA or UPMB in the memory controller whenever you program it. In a read or write transfer, the signal is only asserted if the data lane D[24:31] contains valid data. **PCMCIA Write Enable**—This output signal is asserted when the MPC823 initiates a write access to a memory region controlled by the PCMCIA interface.
$\overline{GPL\_A0}$ $\overline{GPL\_B0}$	E13	**General-Purpose Line 0 on UPMA**—This output signal reflects the value specified in the UPMA in the memory controller when an external transfer to a slave is controlled by the user programmable machine A (UPMA). **General-Purpose Line 0 on UPMB**—This output signal reflects the value specified in the UPMB in the memory controller when an external transfer to a slave is controlled by the user programmable machine B (UPMB).
$\overline{GPL\_A1}$ $\overline{GPL\_B1}$ $\overline{OE}$	C16	**General-Purpose Line 1 on UPMA**—This output signal reflects the value specified in the UPMA in the memory controller when an external transfer to a slave is controlled by the user programmable machine A (UPMA). **General-Purpose Line 1 on UPMB**—This output signal reflects the value specified in the UPMB in the memory controller when an external transfer to a slave is controlled by the user programmable machine B (UPMB). **Output Enable**—This output signal is asserted when the MPC823 initiates a read access to an external slave controlled by the GPCM in the memory controller.

Table 2-1. Signal Descriptions (Continued)

SIGNAL	PIN NUMBER	DESCRIPTION
GPL_A2 GPL_B2 CS2	C15	**General-Purpose Line 2 on UPMA**—This output signal reflects the value specified in the UPMA in the memory controller when an external transfer to a slave is controlled by the user programmable machine A (UPMA). **General-Purpose Line 2 on UPMB**—This output signal reflects the value specified in the UPMB in the memory controller when an external transfer to a slave is controlled by the user programmable machine B (UPMB). **Chip Select 2**—This output signal enables a peripheral or memory device at a programmed address if defined appropriately in the BR6 and OR6 registers of the memory controller.
GPL_A3 GPL_B3 CS3	D14	**General-Purpose Line 3 on UPMA**—This output signal reflects the value specified in the UPMA in the memory controller when an external transfer to a slave is controlled by the user programmable machine A (UPMA). **General-Purpose Line 3 on UPMB**—This output signal reflects the value specified in the UPMB in the memory controller when an external transfer to a slave is controlled by the user programmable machine B (UPMB). **Chip Select 3**—This output signal enables a peripheral or memory device at a programmed address if defined appropriately in the BR6 and OR6 registers of the memory controller.
GPL_A4 UPWAITA AS	D11	**General-Purpose Line 4 on UPMA**—This output signal reflects the value specified in the UPMA in the memory controller when an external transfer to a slave is controlled by the user programmable machine A (UPMA). **User Programmable Machine Wait A**—This input signal is sampled when you need it and when an access to an external slave is controlled by the UPMA in the memory controller. **Address Strobe**—This input pin is driven by an external asynchronous master to indicate a valid address on the A[6:31] lines. The memory controller in the MPC823 will synchronize this signal and control the memory device addressed if it is recognized to be under its control.
GPL_B4 UPWAITB	B13	**General-Purpose Line 4 on UPMB**—This output signal reflects the value specified in the UPMB in the memory controller when an external transfer to a slave is controlled by the user programmable machine B (UPMB). **User Programmable Machine Wait B**—This input signal is sampled when you need it and when an access to an external slave is controlled by the UPMB in the memory controller.
GPL_A5	C12	**General-Purpose Line 5 on UPMA**—This output signal reflects the value specified in the UPMA in the memory controller when an external transfer to a slave is controlled by the user programmable machine A (UPMA). This signal can also be controlled by the UPMB.
PORESET	B3	**Power-On Reset**—When asserted, this input signal causes the MPC823 to enter the power-on reset state.
RSTCONF	C5	**Reset Configuration**—This input signal is sampled by the MPC823 during the assertion of the HRESET signal. If it is asserted, the configuration mode is sampled in the form of the hard reset configuration word driven on the data bus. When this signal is negated, the default configuration mode is adopted by the MPC823. Notice that the initial base address of internal registers is determined in this sequence.
HRESET	B5	**Hard Reset**—This open drain line, when asserted, causes the MPC823 to enter the hard reset state.
SRESET	B4	**Soft Reset**—This open drain line, when asserted, causes the MPC823 to enter the soft reset state.
XTAL	A4	**External Crystal**—This output signal is one of the connections to an external crystal for the internal oscillator circuitry.
EXTAL	A5	**External Crystal**—This signal is one of the connections to an external crystal for the internal oscillator circuitry.
XFC	B2	**External Filter Capacitance**—This input signal is the connection pin to an external capacitor filter for the PLL circuitry.
CLKOUT	D1	**CLKOUT**—This output signal is the clock system frequency.
EXTCLK	A6	**External Clock**—This input signal is the external input clock from an external source.

Table 2-1. Signal Descriptions (Continued)

SIGNAL	PIN NUMBER	DESCRIPTION
TEXP	D5	**Timer Expired**—This output signal reflects the status of the TEXPS bit of the PLPRCR register in the clock interface.
WAIT_B	C4	**Wait Slot B**—This input signal, if asserted low, causes the completion of a transaction to be delayed on the PCMCIA-controlled Slot B.
ALE_B DSCK AT1	B8	**Address Latch Enable B**—This output signal is asserted when the MPC823 initiates an access to a region under the control of the PCMCIA socket B interface. **Development Serial Clock**—This input signal is the clock for the debug port interface. **Address Type 1**—This bidirectional three-state signal is driven by the MPC823 when it initiates a transaction on the external bus. When the transaction is initiated by the internal core, it indicates if the transfer is for problem or privilege state.
IP_B0 IWP0 VFLS0	A8	**Input Port B 0**—This input signal is sensed by the MPC823 and its value and changes are reported in the PIPR and PSCR registers of the PCMCIA interface. **Instruction Watchpoint 0**—This output signal reports the detection of an instruction watchpoint in the program flow executed by the internal core. **Visible History Buffer Flushes Status**—This output signal is output by the MPC823 when you need program instructions flow tracking. It reports the number of instructions flushed from the history buffer in the internal core.
IP_B1 IWP1 VFLS1	C8	**Input Port B 1**—This input signal is sensed by the MPC823 and its value and changes are reported in the PIPR and PSCR registers of the PCMCIA interface. **Instruction Watchpoint 1**—This output signal reports the detection of an instruction watchpoint in the program flow executed by the internal core. **Visible History Buffer Flushes Status**—This output signal is output by the MPC823 when you need program instructions flow tracking. It reports the number of instructions flushed from the history buffer in the internal core.
IP_B2 IOIS16_B AT2	D7	**Input Port B 2**—This input signal is sensed by the MPC823 and its value and changes are reported in the PIPR and PSCR registers of the PCMCIA interface. **I/O Device B is 16 Bits Port Size**—This input signal is monitored by the MPC823 when a PCMCIA interface transaction is initiated to an I/O region in socket B within the PCMCIA space. **Address Type 2**—This bidirectional three-state signal is driven by the MPC823 when it initiates a transaction on the external bus. When the transaction is initiated by the internal core, it indicates if the transfer is instruction or data.
IP_B3 IWP2 VF2	A9	**Input Port B 3**—This input signal is monitored by the MPC823 and its value and changes are reported in the PIPR and PSCR registers of the PCMCIA interface. **Instruction Watchpoint 2**—This output signal reports the detection of an instruction watchpoint in the program flow executed by the internal core. **Visible Instruction Queue Flush Status**—This output signal, together with VF0 and VF1, is output by the MPC823 when you need program instruction flow tracking. VFx reports the number of instructions flushed from the instruction queue in the internal core.
IP_B4 LWP0 VF0	B9	**Input Port B 4**—This input signal is monitored by the MPC823 and its value and changes are reported in the PIPR and PSCR registers of the PCMCIA interface. **Load/Store Watchpoint 0**—This output signal reports the detection of a data watchpoint in the program flow executed by the internal core. **Visible Instruction Queue Flushes Status**—This output signal, together with VF1 and VF2, is output by the MPC823 when you need program instructions flow tracking. VF reports the number of instructions flushed from the instruction queue in the internal core.
IP_B5 LWP1 VF1	C9	**Input Port B 5**—This input signal is monitored by the MPC823 and its value and changes are reported in the PIPR and PSCR registers of the PCMCIA interface. **Load/Store Watchpoint 1**—This output signal reports the detection of a data watchpoint in the program flow executed by the internal core. **Visible Instruction Queue Flushes Status**—This output signal, together with VF0 and VF2, is output by the MPC823 when you need program instructions flow tracking. VF reports the number of instructions flushed from the instruction queue in the internal core.

Table 2-1. Signal Descriptions (Continued)

SIGNAL	PIN NUMBER	DESCRIPTION
IP_B6 DSDI AT0	C7	**Input Port B 6**—This input signal is sensed by the MPC823 and its value and changes are reported in the PIPR and PSCR registers of the PCMCIA interface. **Development Serial Data Input**—This input signal is the data in for the debug port interface. **Address Type 0**—This bidirectional three-state signal is driven by the MPC823 when it initiates a transaction on the external bus. If high (1), the transaction is the CPM. If low (0), the transaction initiator is the core.
IP_B7 $\overline{\text{PTR}}$ AT3	D8	**Input Port B 7**—This input signal is monitored by the MPC823 and its value and changes are reported in the PIPR and PSCR registers of the PCMCIA interface. **Program Trace**—This output signal is asserted by the MPC823 to indicate that an instruction fetch is taking place in order to allow program flow tracking. **Address Type 3**—This bidirectional three-state signal is driven by the MPC823 when it initiates a transaction on the external bus. When the transaction is initiated by the internal core, it indicates if the transfer is reserved for data transfers or a program trace indication for instructions fetch.
MODCK1 OP2 $\overline{\text{STS}}$	D6	**Mode Clock 1**—This input signal is sampled at $\overline{\text{PORESET}}$ negation to configure the PLL/clock mode of operation. **Output Port 2**—This output signal is generated by the MPC823 as a result of a write to the PGCRB register in the PCMCIA interface. **Special Transfer Start**—This output signal is driven by the MPC823 to indicate the beginning of a transaction on the external bus or an internal transaction in show cycle mode.
MODCK2 OP3 DSDO	B6	**Mode Clock 2**—This input signal is sampled at $\overline{\text{PORESET}}$ negation to configure the PLL/clock mode of operation. **Output Port 3**—This output signal is generated by the MPC823 as a result of a write to the PGCRB register in the PCMCIA interface. **Development Serial Data Output**—This output signal is the data out of the debug port interface.
PA[15] USBRXD	P16	**General-Purpose I/O Port A Bit 15**—Bit 15 of the general-purpose I/O port A. **USBRXD**—The receive data input signal for the USB.
PA[14] $\overline{\text{USBOE}}$	R15	**General-Purpose I/O Port A Bit 14**—Bit 14 of the general-purpose I/O port A. **USBOE**—The output enable signal for the USB transmitter.
PA[13] RXD2	R14	**General-Purpose I/O Port A Bit 13**—Bit 13 of the general-purpose I/O port A. **RXD2**—The receive data input signal for the serial communication controller.
PA[12] TXD2	R13	**General-Purpose I/O Port A Bit 12**—Bit 12 of the general-purpose I/O port A. **TXD2**—The transmit data output signal for the serial communication controller. TXD2 has open-drain capability.
PA[9] L1TXDA SMRXD2	N10	**General-Purpose I/O Port A Bit 11**—Bit 9 of the general-purpose I/O port A. **L1TXDA**—The transmit data output signal for the serial interface time-division multiplex port A. This signal has open-drain capability. **SMRXD2**—The serial management controller 2 receive data pin.
PA[8] L1RXDA SMTXD2	T9	**General-Purpose I/O Port A Bit 8**—Bit 8 of the general-purpose I/O port A. **L1RXDA**—The receive data input signal for the serial interface time-division multiplex port A. **SMTXD2**—The serial management controller 2 transmit data pin.
PA[7] CLK1 TIN1 L1RCLKA BRGO1	T8	**General-Purpose I/O Port A Bit 7**—Bit 7 of the general-purpose I/O port A. **CLK1**—This input signal is one of the four clock pins that can be used to clock the serial communication controller, serial management controllers, and USB. **TIN1**—The timer 1 external clock pin. **L1RCLKA**—The receive clock for the serial interface time-division multiplex port A. **BRGO1**—The output clock of BRG1.
PA[6] CLK2 $\overline{\text{TOUT1}}$ TIN3	P8	**General-Purpose I/O Port A Bit 6**—Bit 6 of the general-purpose I/O port A. **CLK2**—This input signal is one of the four clock pins that can be used to clock the serial communication controller, serial management controllers, and USB. **$\overline{\text{TOUT1}}$**—The timer 1 output pin. **TIN3**—The timer 3 external clock pin.

Table 2-1. Signal Descriptions (Continued)

SIGNAL	PIN NUMBER	DESCRIPTION
PA[5] CLK3 TIN2 L1TCLKA BRGO2	T6	**General-Purpose I/O Port A Bit 5**—Bit 5 of the general-purpose I/O port A. **CLK3**—This input signal is one of the four clock pins that can be used to clock the serial communication controller, serial management controllers, and USB. **TIN2**—The timer 2 external clock input pin. **L1TCLKA**—The transmit clock for the serial interface time-division multiplex port A. **BRGO2**—The output clock of BRG2.
PA[4] CLK4 $\overline{\text{TOUT2}}$ TIN4	R6	**General-Purpose I/O Port A Bit 4**—Bit 4 of the general-purpose I/O port A. **CLK4**—This input signal is one of the four clock pins that can be used to clock the serial communication controller, serial management controllers, and USB. **$\overline{\text{TOUT2}}$**—The timer 2 output pin. **TIN4**—The timer 4 external clock pin.
PB[31] $\overline{\text{SPISEL}}$ LCD_A	N14	**General-Purpose I/O Port B Bit 31**—Bit 31 of the general-purpose I/O port B. **$\overline{\text{SPISEL}}$**—The serial peripheral interface slave select input pin. **LCD_A**—This is one of the LCD controller's three extension data bits, which are used to drive an active LCD panel. When using a 12-bit bus instead of a 9-bit bus, the LCD_A signal is the least-significant bit of the red 4-bit code. The red portion of the bus consists of LD[0:2] and LCD_A.
PB[30] SPICLK	P15	**General-Purpose I/O Port B Bit 30**—Bit 30 of the general-purpose I/O port B. **SPICLK**—The serial peripheral interface output clock when it is configured as a master or serial peripheral interface input clock when it is configured as a slave.
PB[29] SPIMOSI	P14	**General-Purpose I/O Port B Bit 29**—Bit 29 of the general-purpose I/O port B. **SPIMOSI**—The serial peripheral interface output data when it is configured as a master or serial peripheral interface input data when it is configured as a slave.
PB[28] SPIMISO BRGO3	T15	**General-Purpose I/O Port B Bit 28**—Bit 28 of the general-purpose I/O port B. **SPIMISO**—The serial peripheral interface input data when it is configured as a master or serial peripheral interface output data when it is configured as a slave. **BRGO3**—The output clock of BRG3.
PB[27] I2CSDA BRGO1	T14	**General-Purpose I/O Port B Bit 27**—Bit 27 of the general-purpose I/O port B. **I2CSDA**—The I^2C serial data pin. This pin is bidirectional and should be configured as an open-drain output. **BRGO1**—The output clock of BRG1.
PB[26] I2CSCL BRGO2	P12	**General-Purpose I/O Port B Bit 26**—Bit 26 of the general-purpose I/O port B. **I2CSCL**—The I^2C serial clock pin. This pin is bidirectional and should be configured as an open-drain output. **BRGO2**—The output clock of BRG2.
PB[25] SMTXD1	N11	**General-Purpose I/O Port B Bit 25**—Bit 25 of the general-purpose I/O port B. **SMTXD1**—The serial management controller 1 transmit data output pin.
PB[24] SMRXD1	T11	**General-Purpose I/O Port B Bit 24**—Bit 24 of the general-purpose I/O port B. **SMRXD1**—The serial management controller 1 receive data input pin.
PB[23] $\overline{\text{SMSYN1}}$ $\overline{\text{SDACK1}}$	T10	**General-Purpose I/O Port B Bit 23**—Bit 23 of the general-purpose I/O port B. **$\overline{\text{SMSYN1}}$**—The serial management controller 1 external sync input pin. **$\overline{\text{SDACK1}}$**—The SDMA acknowledge 1 output pin that is used as a peripheral interface signal for IDMA emulation or as a CAM interface signal for Ethernet.
PB[22] $\overline{\text{SMSYN2}}$ $\overline{\text{SDACK2}}$	R9	**General-Purpose I/O Port B Bit 22**—Bit 22 of the general-purpose I/O port B. **$\overline{\text{SMSYN2}}$**—The serial management controller 2 external sync input pin. **$\overline{\text{SDACK2}}$**—The SDMA acknowledge 2 output pin that is used as a peripheral interface signal for IDMA emulation or as a CAM interface signal for Ethernet.
PB[19] L1ST1 LCD_B	R7	**General-Purpose I/O Port B Bit 19**—Bit 19 of the general-purpose I/O port B. **L1ST1**—One of eight output strobes that can be generated by the serial interface. **LCD_B**—This is one of the LCD controller's three extension data bits, which are used to drive an active LCD panel. When using a 12-bit bus instead of a 9-bit bus, the LCD_B signal is the least-significant bit of the green 4-bit code. The green portion of the bus consists of LD[3:5] and LCD_B.

Table 2-1. Signal Descriptions (Continued)

SIGNAL	PIN NUMBER	DESCRIPTION
PB[18] RTS2 L1ST2	P7	**General-Purpose I/O Port B Bit 18**—Bit 18 of the general-purpose I/O port B. **RTS2**—The Request To Send modem signal for the serial communication controller. **L1ST2**—One of eight output strobes that can be generated by the serial interface.
PB[17] L1ST3 LCD_C	N7	**General-Purpose I/O Port B Bit 17**—Bit 17 of the general-purpose I/O port B. **L1ST3**—One of eight output strobes that can be generated by the serial interface. **LCD_C**—This is one of the LCD controller's three extension data bits, which are used to drive an active LCD panel. When using a 12-bit bus instead of a 9-bit bus, the LCD_C signal is the least-significant bit of the blue 4-bit code. The blue portion of the bus consists of LD[6:8] and LCD_C.
PB[16] L1RQA L1ST4	R5	**General-Purpose I/O Port B Bit 16**—Bit 16 of the general-purpose I/O port B. **L1RQA**—The D-channel request signal for the serial interface time-division multiplex port A. **L1ST4**—One of eight output strobes that can be generated by the serial interface.
PC[15] DREQ1 L1ST5	R16	**General-Purpose I/O Port C Bit 15**—Bit 15 of the general-purpose I/O port C. **DREQ1**—The IDMA channel 1 request input signal. **L1ST5**—One of eight output strobes that can be generated by the serial interface.
PC[14] DREQ2 RTS2 L1ST6	T16	**General-Purpose I/O Port C Bit 14**—Bit 14 of the general-purpose I/O port C. **DREQ2**—The IDMA channel 2 request input signal. **RTS2**—The Request To Send modem signal for the serial communication controller. **L1ST6**—One of eight output strobes that can be generated by the serial interface.
PC[13] L1ST7	P13	**General-Purpose I/O Port C Bit 13**—Bit 13 of the general-purpose I/O port C. **L1ST7**—One of eight output strobes that can be generated by the serial interface.
PC[12] L1RQA L1ST8	T13	**General-Purpose I/O Port C Bit 12**—Bit 12 of the general-purpose I/O port C. **L1RQA**—The D-channel request signal for the serial interface time-division multiplex port A. **L1ST8**—One of eight output strobes that can be generated by the serial interface.
PC[11] USBRXP	R10	**General-Purpose I/O Port C Bit 11**—Bit 11 of the general-purpose I/O port C. **USBRXP**—Used with USBRXN, this signal is used by the USB to detect a single-ended zero and the interconnection speed.
PC[10] TGATE1 USBRXN	P9	**General-Purpose I/O Port C Bit 10**—Bit 10 of the general-purpose I/O port C. **TGATE1**—The timer1/timer2 gate signal. **USBRXN**—Used with USBRXP, this signal is used by the USB to detect a single-ended zero and the interconnection speed.
PC[9] CTS2	R8	**General-Purpose I/O Port C Bit 9**—Bit 9 of the general-purpose I/O port C. **CTS2**—The Clear to Send Modem line for the serial communication controller.
PC[8] CD2 TGATE1	N8	**General-Purpose I/O Port C Bit 8**—Bit 8 of the general-purpose I/O port C. **CD2**—The Carrier Detect Modem line for the serial communication controller. **TGATE1**—The timer1/timer2 gate signal.
PC[7] USBTXP	T5	**General-Purpose I/O Port C Bit 7**—Bit 7 of the general-purpose I/O port C. **USBTXP**—This output signal, in conjunction with USBTXN, are the transmit lines of the USB.
PC[6] USBTXN	N6	**General-Purpose I/O Port C Bit 6**—Bit 6 of the general-purpose I/O port C. **USBTXN**—This output signal, in conjunction with USBTXP, are the transmit lines of the USB.
PC[5] L1TSYNCA SDACK1	P6	**General-Purpose I/O Port C Bit 5**—Bit 5 of the general-purpose I/O port C. **L1TSYNCA**—The transmit sync input for the serial interface time-division multiplex port A. **SDACK1**—The SDMA acknowledge 1output pin that is used as a peripheral interface signal for IDMA emulation or as a CAM interface signal for Ethernet.
PC[4] L1RSYNCA	T4	**General-Purpose I/O Port C Bit 4**—Bit 4 of the general-purpose I/O port C. **L1RSYNCA**—The receive sync input for the serial interface time-division multiplex port A.

Table 2-1. Signal Descriptions (Continued)

SIGNAL	PIN NUMBER	DESCRIPTION
PD[15] LD8 VD7	R4	**General-Purpose I/O Port D Bit 15**—Bit 15 of the general-purpose I/O port D. **LD8**—One of the 12 data bus bits used to drive the LCD panel. **VD7**—One of the data bus bits of the video controller used for driving the video encoder.
PD[14] LD7 VD6	T3	**General-Purpose I/O Port D Bit 14**—Bit 14 of the general-purpose I/O port D. **LD7**—One of the 12 data bus bits used to drive the LCD panel. **VD6**—One of the data bus bits of the video controller used for driving the video encoder.
PD[13] LD6 VD5	P5	**General-Purpose I/O Port D Bit 13**—Bit 13 of the general-purpose I/O port D. **LD6**—One of the 12 data bus bits used to drive the LCD panel. **VD5**—One of the data bus bits of the video controller used for driving the video encoder.
PD[12] LD5 VD4	R3	**General-Purpose I/O Port D Bit 12**—Bit 12 of the general-purpose I/O port D. **LD5**—One of the 12 data bus bits used to drive the LCD panel. **VD4**—One of the data bus bits of the video controller used for driving the video encoder.
PD[11] LD4 VD3	N5	**General-Purpose I/O Port D Bit 11**—Bit 11 of the general-purpose I/O port D. **LD4**—One of the 12 data bus bits used to drive the LCD panel. **VD3**—One of the data bus bits of the video controller used for driving the video encoder.
PD[10] LD3 VD2	T2	**General-Purpose I/O Port D Bit 10**—Bit 10 of the general-purpose I/O port D. **LD3**—One of the 12 data bus bits used to drive the LCD panel. **VD2**—One of the data bus bits of the video controller used for driving the video encoder.
PD[9] LD2 VD1	P4	**General-Purpose I/O Port D Bit 9**—Bit 9 of the general-purpose I/O port D. **LD2**—One of the 12 data bus bits used to drive the LCD panel. **VD1**—One of the data bus bits of the video controller used for driving the video encoder.
PD[8] LD1 VD0	T1	**General-Purpose I/O Port D Bit 8**—Bit 8 of the general-purpose I/O port D. **LD1**—One of the 12 data bus bits used to drive the LCD panel. **VD0**—One of the data bus bits of the video controller used for driving the video encoder.
PD[7] LD0 FIELD	R2	**General-Purpose I/O Port D Bit 7**—Bit 7 of the general-purpose I/O port D. **LD0**—One of the 12 data bus bits used to drive the LCD panel. **FIELD**—The line the video controller uses to signal which of the two fields is the current one.
PD[6] LCD_AC LOE BLANK	R1	**General-Purpose I/O Port D Bit 6**—Bit 6 of the general-purpose I/O port D. **LCD_AC**—This output signal from the LCD controller toggles once every programmable number of frames. It is used with passive panels. **LOE**—The output enable signal that is used with TFT panels. **BLANK**—The video controller uses this signal to let the video encoder know that the current cycle is a blank type.
PD[5] FRAME VSYNC	P2	**General-Purpose I/O Port D Bit 5**—Bit 5 of the general-purpose I/O port D. **FRAME**—The output signal from the video controller that marks the beginning of a new frame. **VSYNC**—The output signal from the LCD controller that marks the beginning of a new frame.
PD[4] LOAD HSYNC	P3	**General-Purpose I/O Port D Bit 4**—Bit 4 of the general-purpose I/O port D. **LOAD**—The output signal from the video controller that marks the beginning of a new display line. **HSYNC**—The output signal from the LCD controller that marks the beginning of a new frame.

Table 2-1. Signal Descriptions (Continued)

SIGNAL	PIN NUMBER	DESCRIPTION
PD[3] SHIFT/CLK CLK	N4	**General-purpose I/O Port D Bit 3**—Bit 3 of the general-purpose I/O port D. **SHIFT/CLK**—This output signal is used to generate the shift clock timing to the LCD panel when using the LCD controller. The direction is defined when you program it. **CLK**—When the video controller is used, the CLK function can either be an output clock to drive the video encoder or an external input clock from the video encoder to drive the video controller. The direction is defined when you program it.
Power Supply	See Table 2-2 for pin breakout.	**VDDL**—Power supply of the internal logic. **VDDH**—Power supply of the I/O buffers and certain parts of the clock control. **VDDSYN**—Power supply of the phase-locked loop circuitry. **VSSSYN**—Power supply of the phase-locked loop ground. **VSSSYN1**—Power supply of the phase-locked loop ground. **GND**—Power supply ground. **KAPWR**—Power supply of the internal oscillator, real-time clock, periodic interrupt timer, decrementer, and timebase.
TCK DSCK	T12	**Test Clock**—This input signal is the clock of the JTAG interface. **Development Serial Clock**—This input signal is the clock for the debug port interface.
TMS	R12	**Test Mode Select**—This input signal controls the TAP machine sequence in the JTAG interface.
TDI DSDI	R11	**Test Data Input**—This input signal is the data in the JTAG interface. **Development Serial Data Input**—This input signal is the data for the debug port interface.
TDO DSDO	N12	**Test Data Output**—This three-state output signal is the data out of the JTAG interface. **Development Serial Data Output**—This output signal is the data out of the debug port interface.
$\overline{\text{TRST}}$	P11	**Test Reset**—This input signal is the asynchronous reset of the TAP machine on the JTAG interface.
N/C	See Table 2-2 for pin breakout.	**No Connect**—These pins are not connected.

Table 2-2. Pin Breakout

	SIGNAL	PIN NUMBER
ADDRESS BUS PINS	A6	M13
	A7	N15
	A8	N16
	A9	M15
	A10	L13
	A11	M16
	A12	M14
	A13	L14
	A14	L15
	A15	L16
	A16	K14
	A17	K13
	A18	G13
	A19	K15
	A20	J15
	A21	J14
	A22	G14
	A23	H15
	A24	H13
	A25	H14
	A26	F14
	A27	K16
	A28	G16
	A29	H16
	A30	G15
	A31	F16

Table 2-2. Pin Breakout (Continued)

	SIGNAL	PIN NUMBER
DATA BUS PINS	D0	M1
	D1	L1
	D2	J2
	D3	J1
	D4	L2
	D5	H1
	D6	F1
	D7	E1
	D8	M2
	D9	K2
	D10	K3
	D11	K1
	D12	M4
	D13	M3
	D14	J3
	D15	J4
	D16	H2
	D17	K4
	D18	H3
	D19	G2
	D20	G3
	D21	F2
	D22	H4
	D23	L4
	D24	F3
	D25	G4
	D26	E4
	D27	L3
	D28	F4
	D29	E2
	D30	D2
	D31	E3

Table 2-2. Pin Breakout (Continued)

	SIGNAL	PIN NUMBER
CHIP SELECT PINS	$\overline{CS0}$	D12
	$\overline{CS1}$	A14
	$\overline{CS2}$	B14
	$\overline{CS3}$	A15
	$\overline{CS4}$	B16
	$\overline{CS5}$	D13
	$\overline{CS6}$	C14
	$\overline{CS7}$	B15
POWER SUPPLY PINS	VDDH	E5–12, F5, F12, G5, G12, H5, H12, J5, J12, K5, K12, L5, L12, M5–M12
	VDDL	A7, G1, J16, T7
	VDDSYN	B1
	KAPWR	A3
	VSSSYN	A1
	VSSSYN1	A2
	GND	F6–F11, G6–G11, H6–H11, J6–J11, K6–K11, L6–L11
NO CONNECT PINS	N/C	A16, C1, C6, E14, J13, N9, N13, P1, P10

SECTION 3
MEMORY MAP

This section discusses the internal memory map (including key registers) of the MPC823. Each memory resource is mapped within a contiguous block of 16K storage. The location of this block within the global 4G real storage space can be mapped on 64K resolution through an implementation specific special register called the internal memory map register (IMMR). Refer to **Section 12.12.1.2 Internal Memory Map Register** for more information.

Note: If you are using Mask Revision Base # F98S, there are only two timers.

Table 3-1. MPC823 Internal Memory Map

INTERNAL ADDRESS	REGISTER	SIZE (IN BITS)	PAGE NUMBER LOCATION
SYSTEM INTERFACE UNIT			
000	SIUMCR—SIU Module Configuration Register	32	**12-30**
004	SYPCR—System Protection Control Register	32	**12-35**
008 to 00D	RES—Reserved	—	—
00E	SWSR—Software Service Register	16	**12-27**
010	SIPEND—SIU Interrupt Pending Register	32	**12-7**
014	SIMASK—SIU Interrupt Mask Register	32	**12-8**
018	SIEL—SIU Interrupt Edge/Level Register	32	**12-9**
01C	SIVEC—SIU Interrupt Vector Register	32	**12-10**
020	TESR—Transfer Error Status Register	32	**12-36**
024 to 02F	RES—Reserved	—	—
030	SDCR—SDMA Configuration Register	32	**16-87**
034 to 07F	RES—Reserved	—	—

Table 3-1. MPC823 Internal Memory Map (Continued)

INTERNAL ADDRESS	REGISTER	SIZE (IN BITS)	PAGE NUMBER LOCATION
PCMCIA			
080	PBR0—PCMCIA Interface Base Register 0	32	**17-16**
084	POR0—PCMCIA Interface Option Register 0	32	**17-17**
088	PBR1—PCMCIA Interface Base Register 1	32	**17-16**
08C	POR1—PCMCIA Interface Option Register 1	32	**17-17**
090	PBR2—PCMCIA Interface Base Register 2	32	**17-16**
094	POR2—PCMCIA Interface Option Register 2	32	**17-17**
098	PBR3—PCMCIA Interface Base Register 3	32	**17-16**
09C	POR3—PCMCIA Interface Option Register 3	32	**17-17**
0A0	PBR4—PCMCIA Interface Base Register 4	32	**17-16**
0A4	POR4—PCMCIA Interface Option Register 4	32	**17-17**
0A8	PBR5—PCMCIA Interface Base Register 5	32	**17-16**
0AC	POR5—PCMCIA Interface Option Register 5	32	**17-17**
0B0	PBR6—PCMCIA Interface Base Register 6	32	**17-16**
0B4	POR6—PCMCIA Interface Option Register 6	32	**17-17**
0B8	PBR7—PCMCIA Interface Base Register 7	32	**17-16**
0BC	POR7—PCMCIA Interface Option Register 7	32	**17-17**
0C0 to 0E3	RES—Reserved	—	—
0E4	PGCRB—PCMCIA Interface General Control Register B	32	**17-15**
0E8	PSCR—PCMCIA Interface Status Change Register	32	**17-11**
0EC to 0EF	RES—Reserved	—	—
0F0	PIPR—PCMCIA Interface Input Pins Register	32	**17-9**
0F4 to 0F7	RES—Reserved	—	—
0F8	PER—PCMCIA Interface Enable Register	32	**17-13**
0FC to 0FF	RES—Reserved	—	—
MEMORY CONTROLLER			
100	BR0—Base Register Bank 0	32	**15-9**
104	OR0—Option Register Bank 0	32	**15-11**
108	BR1—Base Register Bank 1	32	**15-9**
10c	OR1—Option Register Bank 1	32	**15-11**
110	BR2—Base Register Bank 2	32	**15-9**
114	OR2—Option Register Bank 2	32	**15-11**

Table 3-1. MPC823 Internal Memory Map (Continued)

INTERNAL ADDRESS	REGISTER	SIZE (IN BITS)	PAGE NUMBER LOCATION
118	BR3—Base Register Bank 3	32	**15-9**
11C	OR3—Option Register Bank 3	32	**15-11**
120	BR4—Base Register Bank 4	32	**15-9**
124	OR4—Option Register Bank 4	32	**15-11**
128	BR5—Base Register Bank 5	32	**15-9**
12C	OR5—Option Register Bank 5	32	**15-11**
130	BR6—Base Register Bank 6	32	**15-9**
134	OR6—Option Register Bank 6	32	**15-11**
138	BR7—Base Register Bank 7	32	**15-9**
13C	OR7—Option Register Bank 7	32	**15-11**
140 to 163	RES—Reserved	—	—
164	MAR—Memory Address Register	32	**15-27**
168	MCR—Memory Command Register	32	**15-17**
16C to 16F	RES—Reserved	—	—
170	MAMR—Machine A Mode Register	32	**15-19**
174	MBMR—Machine B Mode Register	32	**15-23**
178	MSTAT—Memory Status Register	16	**15-15**
17A	MPTPR—Memory Periodic Timer Prescaler	16	**15-28**
17C	MDR—Memory Data Register	32	**15-26**
180 to 1FF	RES—Reserved	—	—
SYSTEM INTEGRATION TIMERS			
200	TBSCR—Timebase Status and Control Register	16	**12-16**
204	TBREFU—Timebase Reference Register Upper	32	**12-15**
208	TBREFL—Timebase Reference Register Lower	32	**12-15**
20C to 21F	RES—Reserved	—	—
220	RTCSC—Real-Time Clock Status and Control Register	16	**12-18**
224	RTC—Real-Time Clock Register	32	**12-19**
228	RTSEC—Real-Time Clock Alarm Seconds Register	32	**12-20**
22C	RTCAL—Real-Time Clock Alarm Register	32	**12-21**
230 to 23F	RES—Reserved	—	—
240	PISCR—Periodic Interrupt Status and Control Register	16	**12-23**
244	PITC—Periodic Interrupt Timer Count Register	32	**12-24**

Table 3-1. MPC823 Internal Memory Map (Continued)

INTERNAL ADDRESS	REGISTER	SIZE (IN BITS)	PAGE NUMBER LOCATION
248	PITR—Periodic Interrupt Timer Register	32	**12-25**
24C to 27F	RES—Reserved	—	—
CLOCKS AND RESET			
280	SCCR—System Clock and Reset Control Register	32	**5-3**
284	PLPRCR—PLL, Low-Power and Reset Control Register	32	**5-7**
288	RSR—Reset Status Register	32	**4-5**
28C to 2FF	RES—Reserved	—	—
SYSTEM INTEGRATION TIMERS KEYS			
300	TBSCRK—Timebase Status and Control Register Key	32	**5-27**
304	TBREFF0K—Timebase Reference Register 0 Key	32	**5-27**
308	TBREFF1K—Timebase Reference Register 1 Key	32	**5-27**
30C	TBK—Timebase and Decrementer Register Key	32	**5-27**
310 to 31F	RES—Reserved	—	—
320	RTCSCK—Real-Time Clock Status and Control Register Key	32	**5-27**
324	RTCK—Real-Time Clock Register Key	32	**5-27**
328	RTSECK—Real-Time Alarm Seconds Key	32	**5-27**
32C	RTCALK—Real-Time Alarm Register Key	32	**5-27**
330 to 33F	RES—Reserved	—	—
340	PISCRK—Periodic Interrupt Status and Control Register Key	32	**5-27**
344	PITCK—Periodic Interrupt Count Register Key	32	**5-27**
348 to 37F	RES—Reserved	—	—
CLOCKS AND RESET KEYS			
380	SCCRK—System Clock Control Key	32	**5-27**
384	PLPRCRK—PLL, Low Power and Reset Control Register Key	32	**5-27**
388	RSRK—Reset Status Register Key	32	**5-27**
38C to 7FF	RES—Reserved	—	—
VIDEO CONTROLLER			
800	VCCR—Video Controller Configuration Register	16	**19-5**
802 to 803	RES—Reserved	16	—
804	VSR—Video Status Register	8	**19-7**
805	RES—Reserved	8	—

Table 3-1. MPC823 Internal Memory Map (Continued)

INTERNAL ADDRESS	REGISTER	SIZE (IN BITS)	PAGE NUMBER LOCATION
806	VCMR—Video Controller Command Register	8	**19-8**
807	RES—Reserved	8	—
808	VBCB—Video Background Color Buffer Register	32	**19-9**
80C to 80F	RES—Reserved	16	—
810	VFCR0—Video Frame Configuration Register (Set 0)	32	**19-10**
814	VFAA0—Video Frame Buffer A Start Address Register (Set 0)	32	**19-11**
818	VFBA0—Video Frame Buffer B Start Address Register (Set 0)	32	**19-12**
81C	VFCR1—Video Frame Configuration Register (Set 1)	32	**19-13**
820	VFAA1—Video Frame Buffer A Start Address Register (Set 1)	32	**19-14**
824	VFBA1—Video Frame Buffer B Start Address Register (Set 1)	32	**19-15**
828 to 83F	RES—Reserved	—	—
LCD CONTROLLER			
840	LCCR—LCD Panel Configuration Register	32	**18-20**
844	LCHCR—LCD Horizontal Control Register	32	**18-22**
848	LCVCR—LCD Vertical Configuration Register	32	**18-23**
84C to 84F	RES—Reserved	—	—
850	LCFAA—LCD Frame Buffer A Start Address	32	**18-25**
854	LCFBA—LCD Frame Buffer B Start Address	32	**18-26**
858	LCSR—LCD Status Register	8	**18-27**
859 to 85F	RES—Reserved	—	—
I^2C CONTROLLER			
860	I2MOD—I^2C Mode Register	8	**16-458**
864	I2ADD—I^2C Address Register	8	**16-463**
868	I2BRG—I^2C Baud Rate Generator Register	8	**16-464**
86C	I2COM—I^2C Command Register	8	**16-464**
870	I2CER—I^2C Event Register	8	**16-465**
874	I2CMR—I^2C Mask Register	8	**16-466**
875 to 8FF	RES—Reserved	—	—

Table 3-1. MPC823 Internal Memory Map (Continued)

INTERNAL ADDRESS	REGISTER	SIZE (IN BITS)	PAGE NUMBER LOCATION
DMA CONTROLLER			
900 to 903	RES—Reserved	—	—
904	SDAR—SDMA Address Register	32	**16-90**
908	SDSR—SDMA Status Register (DSP Interrupts)	8	**16-88**
909 to 90B	RES—Reserved	—	—
90C	SDMR—SDMA Mask Register (DSP Interrupts)	8	**16-36, 16-89**
90D to 90F	RES—Reserved	—	—
910	IDSR—IDMA1 Status Register	8	**16-95**
911 to 913	RES—Reserved	—	—
914	IDMR—IDMA1 Mask Register	8	**16-96**
915 to 917	RES—Reserved	—	—
918	IDSR—IDMA2 Status Register	8	**16-95**
919 to 91B	RES—Reserved	—	—
91C	IDMR—IDMA2 Mask Register	8	**16-96**
91D to 92F	RES—Reserved	—	—
COMMUNICATIONS PROCESSOR MODULE INTERRUPT CONTROLLER			
930	CIVR—CPM Interrupt Vector Register	16	**16-500**
932 to 93F	RES—Reserved	—	—
940	CICR—CPM Interrupt Configuration Register	32	**16-495**
944	CIPR—CPM Interrupt Pending Register	32	**16-497**
948	CIMR—CPM Interrupt Mask Register	32	**16-498**
94C	CISR—CPM Interrupt In-Service Register	32	**16-499**
PARALLEL PORTS			
950	PADIR—Port A Data Direction Register	16	**16-471**
952	PAPAR—Port A Pin Assignment Register	16	**16-471**
954	PAODR—Port A Open-Drain Register	16	**16-470**
956	PADAT—Port A Data Register	16	**16-470**
958 to 95F	RES—Reserved	—	—
960	PCDIR—Port C Data Direction Register	16	**16-482**
962	PCPAR—Port C Pin Assignment Register	16	**16-483**
964	PCSO—Port C Special Options Register	16	**16-483**

Table 3-1. MPC823 Internal Memory Map (Continued)

INTERNAL ADDRESS	REGISTER	SIZE (IN BITS)	PAGE NUMBER LOCATION
966	PCDAT—Port C Data Register	16	**16-482**
968	PCINT—Port C Interrupt Control Register	16	**16-484**
96A to 96F	RES—Reserved	—	—
970	PDDIR—Port D Data Direction Register	16	**16-486**
972	PDPAR—Port D Pin Assignment Register	16	**16-487**
974	RES—Reserved	—	—
976	PDDAT—Port D Data Register	16	**16-486**
978 to 97F	RES—Reserved	—	—
CPM TIMERS			
980	TGCR—Timer Global Configuration Register	16	**16-79**
982 to 98F	RES—Reserved	—	—
990	TMR1—Timer1 Mode Register	16	**16-80**
992	TMR2—Timer2 Mode Register	16	**16-80**
9A0	TMR3—Timer3 Mode Register	16	**16-80**
9A2	TMR4—Timer4 Mode Register	16	**16-80**
994	TRR1—Timer1 Reference Register	16	**16-81**
996	TRR2—Timer2 Reference Register	16	**16-81**
9A4	TRR3—Timer3 Reference Register	16	**16-81**
9A6	TRR4—Timer4 Reference Register	16	**16-81**
998	TCR1—Timer1 Capture Register	16	**16-82**
99A	TCR2—Timer2 Capture Register	16	**16-82**
9A8	TCR3—Timer3 Capture Register	16	**16-82**
9AA	TCR4—Timer4 Capture Register	16	**16-82**
99C	TCN1—Timer1 Counter Register	16	**16-82**
99E	TCN2—Timer2 Counter Register	16	**16-82**
9AC	TCN3—Timer3 Counter Register	16	**16-82**
9AE	TCN4—Timer4 Counter Register	16	**16-82**
9A0 to 9AF	RES—Reserved	—	—
9B0	TER1—Timer1 Event Register	16	**16-83**
9B2	TER2—Timer2 Event Register	16	**16-83**

Table 3-1. MPC823 Internal Memory Map (Continued)

INTERNAL ADDRESS	REGISTER	SIZE (IN BITS)	PAGE NUMBER LOCATION
9B4	TER3—Timer3 Event Register	16	**16-83**
9B6	TER4—Timer4 Event Register	16	**16-83**
9B4 to 9BF	RES—Reserved	—	—
COMMUNICATION PROCESSOR MODULE			
9CO	CPCR—Communication Processor Module Command Register	16	**16-10**
9C2 to 9C3	RES—Reserved	16	—
9C4	RCCR—RISC Controller Configuration Register	16	**16-8**
9C6	RES—Reserved	—	—
9C7	RMDS—RISC Microcode Development Support Control Register	8	**16-7**
9D6	RTER—RISC Timer Event Register	16	**16-25**
9DA	RTMR—RISC Timer Mask Register	16	**16-25**
9DC to 9EF	RES—Reserved	—	—
BAUD RATE GENERATORS			
9F0	BRGC1—BRG1 Configuration Register	32	**16-154**
9F4	BRGC2—BRG2 Configuration Register	32	**16-154**
9F8	BRGC3—BRG3 Configuration Register	32	**16-154**
9FC	BRGC4—BRG4 Configuration Register	32	**16-154**
UNIVERSAL SERIAL BUS			
A00	USMOD—USB Mode Register	8	**16-357**
A01	USADR—USB Slave Address Register	8	**16-363**
A02	USCOM—USB Command Register	8	**16-364**
A03	RES—Reserved	8	—
A04	USEP0—USB End Point 0 Register	16	**16-365**
A06	USEP1—USB End Point 1 Register	16	**16-365**
A08	USEP2—USB End Point 2 Register	16	**16-365**
A0A	USEP3—USB End Point 3 Register	16	**16-365**
A0C to A0F	RES—Reserved	—	—
A10	USBER—USB Event Register	16	**16-368**
A12	RES—Reserved	16	—
A14	USBMR—USB Mask Register	16	**16-369**
A16	RES—Reserved	8	—

Table 3-1. MPC823 Internal Memory Map (Continued)

INTERNAL ADDRESS	REGISTER	SIZE (IN BITS)	PAGE NUMBER LOCATION
A17	USBS—USB Status Register	8	**16-369**
A18 to A1F	RES—Reserved	—	—
SERIAL COMMUNICATION CONTROLLER			
A20	GSMR_L—SCC2 General Mode Low Register	32	**16-160**
A24	GSMR_H—SCC2 General Mode High Register	32	**16-160**
A28	SCC2 PSMR—Protocol-Specific Mode Register	16	**16-170** **16-211** (UART) **16-236** (HDLC) **16-272** (AHDLC) **16-302** (Trans)
A2A to A2B	RES—Reserved	16	—
A2C	TODR—Transmit-on-Demand Register	16	**16-171**
A2E	DSR—Data Synchronization Register	16	**16-171**
A30	SCCE—SCC2 Event Register	16	**16-181** **16-221** (UART) **16-243** (HDLC) **16-277** (AHDLC) **16-307** (Trans)
A32	RES—Reserved	16	—
A34	SCCM—SCC2 Mask Register	16	**16-181** **16-223** (UART) **16-246** (HDLC) **16-309** (Trans)
A36	RES—Reserved	8	—
A37	SCCS—SCC2 Status Register	8	**16-181** **16-224** (UART) **16-247** (HDLC) **16-309** (Trans)
A38	IRMODE—SCC2 Infra-Red Mode Register	16	**16-287**
A3A	IRSIP—SCC2 Infra-Red Serial Interaction Pulse Control Register	16	**16-289**
A3C to A81	RES—Reserved	—	—
SERIAL MANAGEMENT CONTROLLER 1			
A82	SMCMR—SMC Mode Register	16	**16-374** **16-388** (UART) **16-406** (Trans) **16-421** (GCI)
A84	RES—Reserved	16	—
A86	SMCE—SMC Event Register	8	**16-395** (UART) **16-412** (Trans) **16-422** (GCI)
A87 to A89	RES—Reserved	—	—

Table 3-1. MPC823 Internal Memory Map (Continued)

INTERNAL ADDRESS	REGISTER	SIZE (IN BITS)	PAGE NUMBER LOCATION
A8A	SMCM—SMC Mask Register	8	**16-397** (UART) **16-413** (Trans) **16-423** (GCI)
A8B to A91	RES—Reserved	—	—
SERIAL MANAGEMENT CONTROLLER 2			
A92	SMCMR—SMC Mode Register	16	**16-374** **16-388** (UART) **16-406** (Trans) **16-421** (GCI)
A94	RES—Reserved	16	—
A96	SMCE—SMC Event Register	8	**16-395** (UART) **16-412** (Trans) **16-422** (GCI)
A97 to A99	RES—Reserved	—	—
A9A	SMCM—SMC Mask Register	8	**16-397** (UART) **16-413** (Trans) **16-423** (GCI)
A9B to A9F	RES—Reserved	—	—
SERIAL PERIPHERAL INTERFACE			
AA0	SPMODE—SPI Mode Register	16	**16-433**
AA2	RES—Reserved	16	—
AA6	SPIE—SPI Event Register	8	**16-442**
AA7 to AA9	RES—Reserved	—	—
AAA	SPIM—SPI Mask Register	8	**16-443**
AAB	RES—Reserved	16	—
AAD	SPCOM—SPI Command Register	8	**16-441**
AAE to AB1	RES—Reserved	—	—
PORT B			
AB8	PBDIR—Port B Data Direction Register	32	**16-477**
ABC	PBPAR—Port B Pin Assignment Register	32	**16-478**
AC0	PBODR—Port B Open-Drain Register	32	**16-475**
AC4	PBDAT—Port B Data Register	32	**16-476**

Table 3-1. MPC823 Internal Memory Map (Continued)

INTERNAL ADDRESS	REGISTER	SIZE (IN BITS)	PAGE NUMBER LOCATION
SERIAL INTERFACE			
AE0	SIMODE—Serial Interface Mode Register	32	**16-127**
AE4	SIGMR—Serial Interface Global Mode Register	8	**16-126**
AE5	RES—Reserved	8	—
AE6	SISTR—Serial Interface Status Register	8	**16-137**
AE7	SICMR—Serial Interface Command Register	8	**16-136**
AE8 to AEB	RES—Reserved	—	—
AEC	SICR—Serial Interface Clock Route Register	32	**16-134**
AF0	SIRP—Serial Interface RAM Pointer Register	32	**16-137**
AF4 to AFF	RES—Reserved	—	—
SPECIALIZED RAM			
B00 to BFF	VCRAM—Video Controller RAM Array	256 bytes	**19-16**
C00 to DFF	SIRAM—Serial Interface RAM	512 bytes	**16-120**
E00 to FFF	LCOLR—LCD Color RAM	512 bytes	**18-28**
1000 to 1FFF	RES—Reserved	—	—
DUAL-PORT RAM			
2000 to 2FFF	DPRAM—Dual-Port RAM	4,096 bytes	—
3000 to 3BFF	DPRAM—Dual-Port RAM Expansion	—	—
3C00 to 3FFF	PRAM—Parameter RAM	1,024 bytes	—

SECTION 4
RESET

The reset block of the MPC823 has a reset control logic that determines the cause of reset, synchronizes it if necessary, and resets the appropriate logic modules. The memory controller, system protection logic, interrupt controller, and parallel I/O pins are initialized only on hard reset. Soft reset initializes the internal logic while maintaining the system configuration.

Table 4-1. Possible Reset Results

RESET SOURCE	RESET EFFECT						
	RESET LOGIC AND PLL STATE RESET	SYSTEM CONFIG RESET	CLOCK MODULE RESET	$\overline{\text{HRESET}}$ PIN DRIVEN	DEBUG PORT CONFIG	OTHER INTERNAL LOGIC RESET	$\overline{\text{SRESET}}$ PIN DRIVEN
Power-On Reset	√	√	√	√	√	√	√
External Hard Reset Loss-of-Lock Software Watchdog Check Stop Debug Port Hard Reset JTAG Reset	—	√	√	√	√	√	√
External Soft Reset Debug Port Soft Reset	—	√	—	—	√	√	√

NOTE: √ indicates that the logic circuitry is reset or the appropriate pin is driven by the source.
— indicates that the logic circuitry is not affected.

4.1 TYPES OF RESET

The MPC823 has several types of inputs to the reset logic:

- Power-on reset
- External hard reset
- Internal hard reset
  - Loss of lock
  - Software watchdog reset
  - Checkstop reset
  - Debug port hard reset
  - JTAG reset
- External soft reset
- Internal soft reset
  - Debug port soft reset
  - JTAG soft reset

All of these reset sources are fed into the reset controller and, depending on the source of the reset, different actions are taken. The reset status register reflects the last source to cause a reset.

4.1.1 Power-On Reset

$\overline{\text{PORESET}}$ (power-on reset) is an active low input pin. In a system with power-down low-power mode, this pin should only be activated when a voltage in the keep-alive power (KAPWR) rail fails. When this pin is asserted, the MODCK bits are sampled and the phase-locked loop multiplication factor and pitrtclk and tmbclk sources are changed to their default values. When this pin is negated, internal MODCK values are unchanged. The $\overline{\text{PORESET}}$ pin should be asserted for a minimum of 3 microseconds. After detecting this assertion, the MPC823 enters the power-on reset state and stays there until the following events occur:

- The internal PLL enters the lock state and the system clock is active
- The $\overline{\text{PORESET}}$ pin is negated

When $\overline{\text{PORESET}}$ is asserted, the MPC823 enters the power-on reset (POR) state in which $\overline{\text{SRESET}}$ and $\overline{\text{HRESET}}$ are asserted by the core. When the MPC823 remains in POR, the extension counter of 512 is reset, and the MODCK pins are sampled when POR pin is negated. After the negation of $\overline{\text{PORESET}}$ or PLL lock, the core enters the state of internal initiated $\overline{\text{HRESET}}$ and continues driving the $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ pins for 512 cycles. When the timer expires, which is usually after the 512 cycles, the configuration is sampled from the data pins and the core stops driving the pins. An external pull-up resistor should drive the $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ pins high. After the pins are negated, a 16-cycle period passes before the presence of an external (hard/soft) reset is tested. Refer to **Section 4.3.1 Hard Reset** for more information.

4.1.2 External Hard Reset

$\overline{\text{HRESET}}$ (hard reset) is a bidirectional, active low I/O pin. The MPC823 can only detect an external assertion of $\overline{\text{HRESET}}$ if it occurs while the MPC823 is not asserting reset. During $\overline{\text{HRESET}}$, $\overline{\text{SRESET}}$ is asserted. $\overline{\text{HRESET}}$ is an open-collector type of pin. $\overline{\text{SRESET}}$ (soft reset) is a bidirectional, active low I/O pin. The MPC823 can only detect an external assertion of $\overline{\text{SRESET}}$ if it occurs while the MPC823 is not asserting reset. The $\overline{\text{SRESET}}$ is also an open-collector type of pin.

When an external $\overline{\text{HRESET}}$ is asserted, the core starts driving the $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ for 512 cycles. When the timer expires, after 512 cycles, the configuration is sampled from the data pins and the core stops driving the $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ pins. An external pull-up resistor should drive the pins high and once they are negated, a 16-cycle period passes before the presence of an external (hard/soft) reset is tested. Refer to **Section 4.3.1 Hard Reset** for more information.

4.1.3 Internal Hard Reset

When the core finds a reason to assert $\overline{\text{HRESET}}$, it starts driving the $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ pins for 512 cycles. When the timer expires, after the 512 cycles, the configuration is sampled from data pins and the core stops driving the pins. An external pull-up resistor should drive the $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ pins high and once they are negated a 16-cycle period passes before the presence of an external (hard/soft) reset is tested. Refer to **Section 4.3.1 Hard Reset** for more information. The causes of internal hard reset are as follows:

- Loss of lock
- Software watchdog reset
- Checkstop reset
- Debug port hard reset
- JTAG reset

4.1.3.1 LOSS OF LOCK. If the PLL detects a loss of lock, erroneous external bus operation occurs if synchronous external devices use the core input clock. Erroneous operation could also occur if devices with a PLL use the core clockout. This source of reset can be asserted if the LOLRE bit in the PLL low-power and reset control register is set. The enabled PLL loss-of-lock event generates an internal hard reset sequence.

4.1.3.2 SOFTWARE WATCHDOG RESET. After the core watchdog counts to zero, a software watchdog reset is asserted. The enabled software watchdog event then generates an internal hard reset sequence.

4.1.3.3 CHECKSTOP RESET. If the core enters a checkstop state and the checkstop reset is enabled, the checkstop reset is asserted. The enabled checkstop event then generates an internal hard reset sequence.

4.1.3.4 DEBUG PORT HARD RESET. When the development port receives a hard reset request from the development tool, an internal hard reset sequence is generated. In this case, the development tool must reconfigure the debug port. See **Section 20.2.1.2.6 Detecting the Trace Window End Address** for more information.

4.1.3.5 JTAG RESET. When the JTAG logic asserts the JTAG soft reset signal, an internal soft reset sequence will be generated.

4.1.4 External Soft Reset

When an external $\overline{\text{SRESET}}$ is asserted, the core starts driving the $\overline{\text{SRESET}}$ pin. When the timer expires, after 512 cycles, the debug port configuration is sampled from the DSDI and DSCK pins and the core stops driving the pin. An external pull-up resistor should drive it high and once it is negated a 16-cycle period passes before the presence of an external soft reset is tested.

4.1.5 Internal Soft Reset

When the core finds a reason to assert $\overline{\text{SRESET}}$, it starts driving the $\overline{\text{SRESET}}$ pin. When the timer expires, after 512 cycles, the debug port configuration is sampled from the DSDI and DSCK pins and the core stops driving the $\overline{\text{SRESET}}$ pin. An external pull-up resistor should drive the pin high and once it is negated a 16-cycle period passes before the presence of an external soft reset is tested. JTAG and the debug port cause an internal soft reset.

4.1.5.1 DEBUG PORT SOFT RESET. When the development port receives a soft reset request from the development tool, an internal soft reset sequence is generated. In this case the development tool must reconfigure the debug port. See **Section 20.2.1.2.6 Detecting the Trace Window End Address** for more information.

4.2 RESET STATUS REGISTER

The 32-bit reset status register (RSR) is powered by the keep-alive power supply. As shown in **Section 3 Memory Map**, it is memory-mapped into the MPC823 system interface unit register map and receives its default reset values at power-on reset.

RSR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	EHRS	ESRS	LLRS	SWRS	CSRS	DBHRS	DBSRS	JTRS	RESERVED							
RESET	0	0	0	0	0	0	0	0	0							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W							
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED															
RESET	0															
R/W	R/W															

EHRS—External Hard Reset Status

This bit is cleared by a power-on reset. When an external hard reset event is detected, this bit is set and remains that way until the software clears it. The EHRS bit can be negated by writing a 1, but a write of zero has no effect on it.

0 = No external hard reset event occurred.
1 = An external hard reset event occurred.

ESRS—External Soft Reset Status

This bit is cleared by a power-on reset. When an external soft reset event is detected, this bit is set and remains that way until the software clears it. The ESRS bit can be negated by writing a 1, but a write of zero has no effect on it.

0 = No external soft reset event occurred.
1 = An external soft reset event occurred.

LLRS—Loss-of-Lock Reset Status

This bit is cleared by a power-on reset. When a loss-of-lock event is enabled by the LOLRE bit in the PLPRCR is detected, this bit is set and remains that way until the software clears it. The LLRS bit can be negated by writing a 1, but a write of zero has no effect on it.

0 = No enabled loss-of-lock reset event occurred.
1 = An enabled loss-of-lock reset event occurred.

SWRS—Software Watchdog Reset Status

This bit is cleared by a power-on reset. When a software watchdog expire event occurs, this bit is set and remains that way until the software clears it. The SWRS bit can be negated by writing a 1, but a write of zero has no effect on it.

0 = No software watchdog reset event occurred.
1 = A software watchdog reset event occurred.

CSRS—Check Stop Reset Status

This bit is cleared by a power-on reset. When the core enters the checkstop state and the checkstop reset is enabled by the CSR bit in the PLPRCR, this bit is set and remains that way until the software clears it. The CSRS bit can be negated by writing a 1, but a write of zero has no effect on it.

0 = No enabled checkstop reset event occurred.
1 = An enabled checkstop reset event occurred.

DBHRS—Debug Port Hard Reset Status

This bit is cleared by a power-on reset. When the debug port hard reset request is set, this bit is set and remains that way until the software clears it. The DBHRS bit can be negated by writing a 1, but a write of zero has no effect on it.

0 = No debug port hard reset request occurred.
1 = A debug port hard reset request occurred.

DBSRS—Debug Port Soft Reset Status

This bit is cleared by a power-on reset. When the debug port soft reset request is set, this bit is set and remains that way until the software clears it. The DBSRS bit can be negated by writing a 1, but a write of zero has no effect on it.

0 = No debug port soft reset request occurred.
1 = A debug port soft reset request occurred.

JTRS—JTAG Reset Status

This bit is cleared by a power-on reset. When the JTAG reset request is set, this bit is set and remains that way until the software clears it. The JTRS bit can be negated by writing a 1, but a write of zero has no effect on it.

0 = No JTAG reset event occurred.
1 = A JTAG reset event occurred.

Bits 8–31—Reserved

These bits are reserved and should be set to 0.

4.3 HOW TO CONFIGURE RESET

In normal operation, you can configure reset with a hard reset. However, to configure the development port you should use a soft reset.

4.3.1 Hard Reset

When a hard reset event occurs, the MPC823 reconfigures its hardware system as well as the development port configuration. The logical value of the bits that determine its initial mode of operation are sampled either from the data bus or from an internal default constant (D[0:31]=x'00000000). If, at sampling time, $\overline{\text{RSTCONF}}$ is asserted, the configuration is sampled from the data bus. Otherwise, it is sampled from the internal default. While $\overline{\text{HRESET}}$ and $\overline{\text{RSTCONF}}$ are asserted, the MPC823 pulls the data bus low through a weak resistor. You can overwrite this default by driving high to the appropriate bit, as shown in Figure 4-1. Figures 4-2 through 4-4 illustrate how reset configuration works when PORESET is asserted. While the $\overline{\text{PORESET}}$ input signal is being asserted, the core assumes the default reset configuration that changes when $\overline{\text{PORESET}}$ is negated or the CLKOUT signal starts oscillating. In this last case, the hardware configuration is sampled every nine clock cycles on the rising edge of the CLKOUT. The setup time required for the data bus is 15 cycles and the maximum rise time of $\overline{\text{HRESET}}$ should be less than six clock cycles. For more information, see **Section 4.3.2 Soft Reset**.

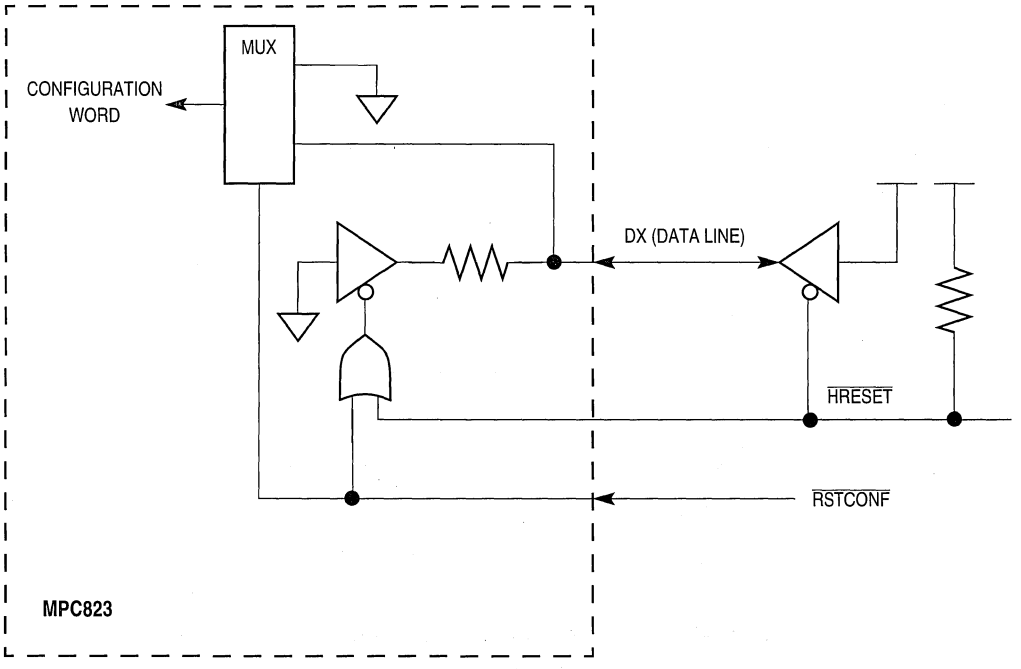


Figure 4-1. Reset Configuration Basic Scheme

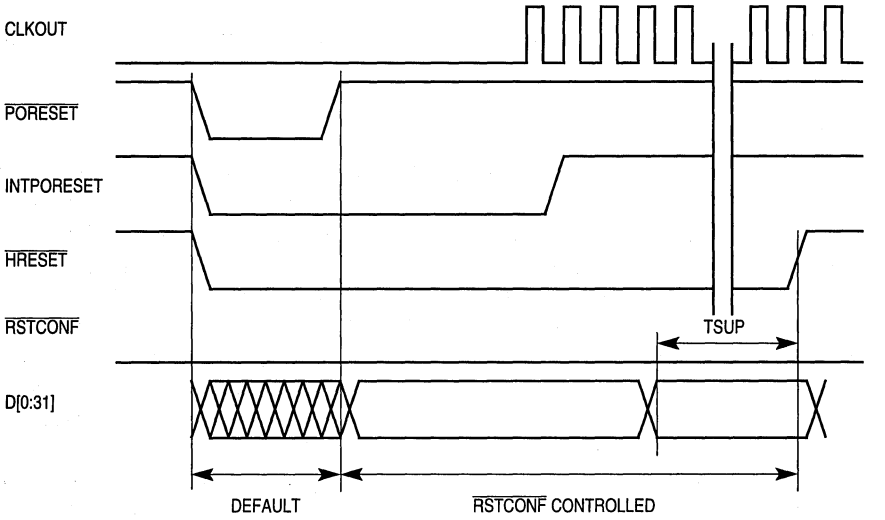


Figure 4-2. Reset Configuration Sampling Scheme For Short PORESET Assertion

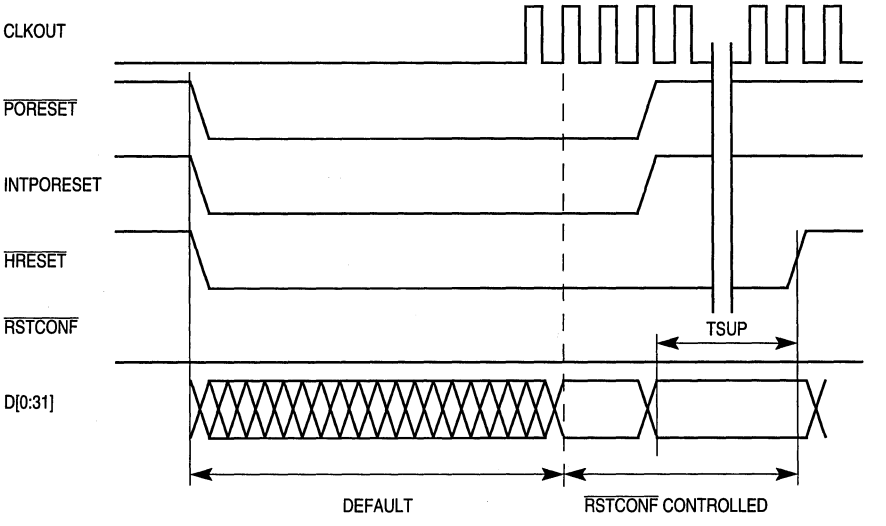


Figure 4-3. Reset Configuration Sampling Scheme For Long PORESET Assertion

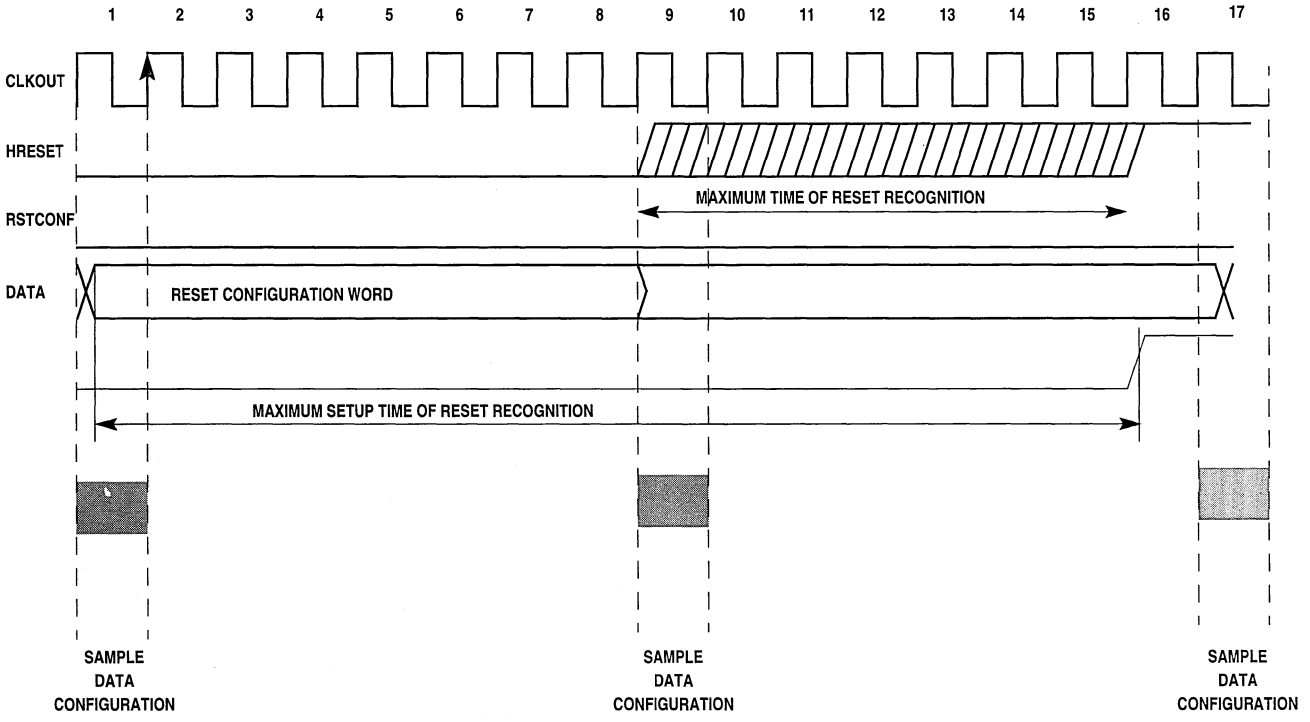


Figure 4-4. Reset Configuration Sampling Timing Requirements

4.3.1.1 HARD RESET CONFIGURATION WORD. The hard reset configuration word is sampled from the data bus. At reset, the bits will determine the default values of the corresponding bits in the SIUMCR, IMMR, and MSR.

HARD RESET CONFIGURATION WORD

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	EARB	IIP	RES	BDIS	BPS		RES	ISB		DBGC		DBPC		EBDF		RES
DEFAULT	0	0	0	0	0		0	0		0		0		0		0
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED															
DEFAULT	0															

NOTE: The default value is due to the internal pull-down resistor on the data bus.

EARB—External Arbitration

If this bit is set (1), external arbitration is assumed. If it is cleared (0), then internal arbitration is performed. See **Section 12 System Interface Unit** for more information.

$\overline{IIP}$—Initial Interrupt Prefix

This bit defines the initial value of the MSR_{IP} immediately after reset. The MSR_{IP} bit defines the interrupt table location. If $\overline{IIP}$ is zero (default), the MSR_{IP} initial value is one, but if it is sampled one, the MSR_{IP} initial value is zero.

Bits 2, 6, and 15—Reserved

These bits are reserved and should be left open.

BDIS—Boot Disable

0 = The memory controller is activated after reset so that it matches all addresses.
1 = The memory controller is not activated after reset, but it is cleared.

BPS—Boot Port Size

This field defines the port size of the boot device.

00 = 32-bit port size.
01 = 8-bit port size.
10 = 16-bit port size.
11 = Reserved.

ISB—Initial Internal Space Base Select

This field defines the initial value of the IMMR bits 0-15 and determines the base address of the internal memory space.

00 = 0x00000000.
01 = 0x00F00000.
10 = 0xFF000000.
11 = 0xFFF00000.

DBGC—Debug Pins Configuration

This field configures the functionality of the following pins.

0x = IWP[0:1]/VFLS[0:1] functions as IWP[0:1].
IWP2/VF2 functions as IWP2.
LWP0/VF0 functions as LWP0.
LWP1/VF1 functions as LWP1.
MODCK1/$\overline{\text{STS}}$ functions as $\overline{\text{STS}}$.
DSCK/AT1 functions as AT1.
DSDI/$\overline{\text{IRQ5}}$ functions as $\overline{\text{IRQ5}}$.
$\overline{\text{PTR}}$/AT3 functions as AT3.
MODCK2/DSDO functions as DSDO.

10 = Reserved.

11 = IWP[0:1]/VFLS[0:1] functions as VFLS[0:1].
IWP2/VF2 functions as VF2.
LWP0/VF0 functions as VF0.
LWP1/VF1 functions as VF1.
MODCK1/$\overline{\text{STS}}$ functions as $\overline{\text{STS}}$.
DSCK/AT1 functions as AT1.
DSDI/$\overline{\text{IRQ5}}$ functions as $\overline{\text{IRQ5}}$.
$\overline{\text{PTR}}$/AT3 functions as AT3.
MODCK2/DSDO functions as DSDO.

DBPC—Debug Port Pins Configuration

This field configures the following pins on the active development port.

00 = DSCK/AT1 functions as defined by DBGC.
DSDI/$\overline{\text{IRQ5}}$ functions as defined by DBGC.
$\overline{\text{PTR}}$/AT3 functions as defined by DBGC.
TCK/DSCK functions as DSCK.
TDI/DSDI functions as DSDI.
TDO/DSDO functions as DSDO.

01 = DSCK/AT3 functions as defined by DBGC.
DSDI/$\overline{\text{IRQ5}}$ functions as defined by DBGC.
$\overline{\text{PTR}}$/AT3 functions as defined by DBGC.
TCK/DSCK functions as TCK.
TDI/DSDI functions as TDI.
TDO/DSDO functions as TDO.

10 = Reserved.

11 = DSCK/AT1 functions as DSCK.
DSDI/$\overline{\text{IRQ5}}$ functions as DSDI.
$\overline{\text{PTR}}$/AT3 functions as $\overline{\text{PTR}}$.
TCK/DSCK functions as TCK.
TDI/DSDI functions as TDI.
TDO/DSDO functions as TDO.

EBDF—External Bus Division Factor

These bits define the frequency division factor between GCLK1/GCLK2 and GCLK1_50/GCLK2_50. CLKOUT is similar to GCLK2_50. GCLK2_50 and GCLK1_50 are used by the system interface unit and memory controller to interface with the external system. The EBDF bits are initialized during $\overline{\text{HRESET}}$ using the hard reset configuration mechanism.

4.3.2 Soft Reset

When a soft reset event occurs, the MPC823 reconfigures the development port.

SECTION 5
CLOCKS AND POWER CONTROL

The MPC823 clock system provides many different timing options for all on-chip and external devices. It contains phase-locked loop circuitry and frequency dividers that generate programmable clock timing for baud rate generators, timers, the LCD controller, and a variety of low-power mode options.

The programmable phase-locked loop, called the *system* phase-locked loop (SPLL) in the MPC823, generates the overall system operating frequency. You can program the SPLL in integer multiples of the input clock frequency. The minimum internal operating frequency is 15MHz. To generate the system operating frequencies, divide by a power of two divider. The clock sources to the timebase, decrementer, real-time counter, and periodic interrupt counter are generated by the MPC823 clock module. For additional timer information, refer to **Section 12 System Interface Unit**.

The MPC823 has a variety of programmable modes that allow your system to operate at its highest level, and yet it still gives you the option of operating in a power-saving mode. Figure 5-1 illustrates the internal clock source and distribution that includes the SPLL, clock dividers, drivers, and main clock oscillator.

5.1 FEATURES

The following list summarizes the main features of the MPC823 clocks and power control system:

- Contains System Phase-Locked Loop (SPLL)
- Clock Dividers Are Provided for Low-Power Modes and Internal Clocks
- Contains Five Major Power-Saving Modes:
  - Normal, Doze, Sleep, Deep Sleep, and Power-Down. Normal and Doze have both high and low modes of operation.
- Able to Operate the Core at Low Voltage for Various Power-Saving Modes

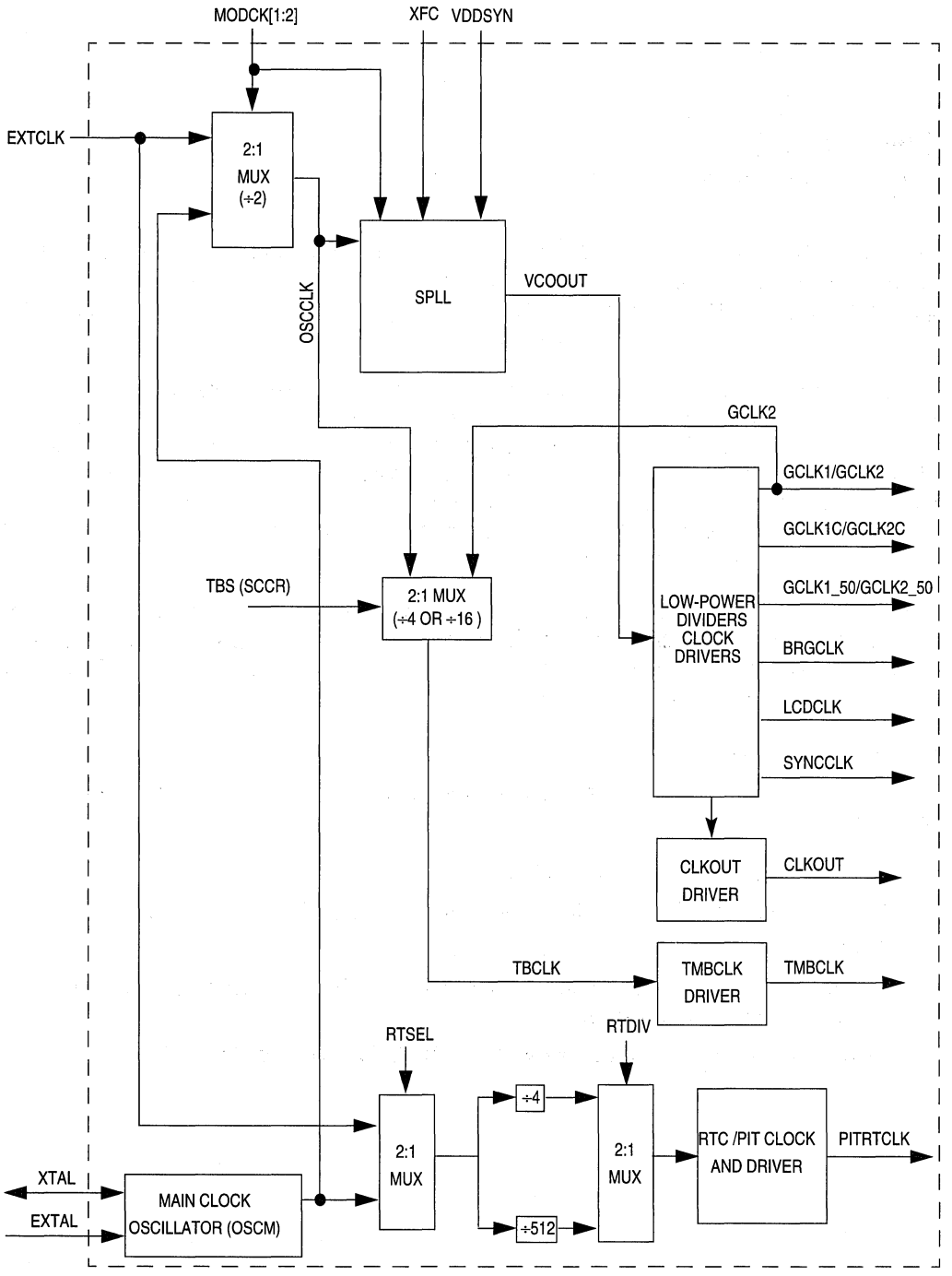


Figure 5-1. Clock Source and Distribution

5.2 REGISTER MODEL

5.2.1 System Clock and Reset Control Register

The SPLL has a 32-bit control register that is powered by keep-alive power. The system clock and reset control register (SCCR) is memory-mapped into the MPC823 system interface unit's register map.

SCCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RES	COM		RESERVED			TBS	RTDIV	RTSEL	CRQEN	PRQEN	RESERVED		EBDF		RES
HRESET	–	#		0			#	#	#	0	0	0		†		0
POR	0	0		0			0	*	*	0	0	0		†		0
R/W	R/W	R/W		R/W			R/W	R/W	R/W	R/W	R/W	R/W		R/W		R/W
ADDR	(IMMR & 0xFFFF0000) + 0x280															

BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RES	DFSYNC		DFBRG		DFNL			DFNH			DFLCD			DFALCD	
HRESET	0	0		0		0			0			0			0	
POR	0	0		0		0			0			0			0	
R/W	R/W	R/W		R/W		R/W			R/W			R/W			R/W	
ADDR	(IMMR & 0xFFFF0000) + 0x282															

NOTE: HRESET is hard reset and POR is power-on reset.
– = undefined and # = unaffected.
* RTDIV depends on the combination of MODCK1and MODCK2. RTSEL depends on MODCK1. See Table 5-2 for more information.
† This field is set according to the default of the hard reset configuration word.

Bits 0 and 3–5—Reserved

These bits are reserved and should be set to 0.

COM—Clock Output Module

This field controls the output buffer strength of the CLKOUT pin. When both bits are set, the CLKOUT pin is held in the high state. These bits can be dynamically changed without generating spikes on the CLKOUT pin. If the CLKOUT pin is not connected to external circuits, both bits are should be set to minimize noise and power dissipation. The COM field is cleared by hard reset.

00 = Clock output enabled full-strength buffer.
01 = Clock output enabled half-strength output buffer.
10 = Reserved.
11 = Clock output disabled.

TBS—Timebase Source

This bit determines the clock source that drives the timebase and decrementer.

0 = Timebase frequency source is OSCCLK divided by 4 or 16.
1 = Timebase frequency source is GCLK2 divided by 16.

RTDIV—Real-Time Clock Divide

This bit indicates if the clock, the crystal oscillator or main clock oscillator, to the real-time clock and periodic interrupt timer is divided by 4 or 512. At power-on reset this bit is cleared if the MODCK1 and MODCK2 signals are low.

0 = The clock is divided by 4.
1 = The clock is divided by 512.

RTSEL—Real-Time Clock Select

This bit selects the crystal oscillator or main clock oscillator as the input source to the real-time clock. At power-on reset, this bit reflects the value of the MODCK1 signal.

0 = The main clock oscillator is selected.
1 = The crystal oscillator is selected.

CRQEN—CPM Request Enable

This bit is cleared by power-on or hard reset and specifies if the general system clock returns to the high frequency while the communication processor module's RISC microcontroller is active.

0 = The system remains in the lower frequency even if the communication processor module is active.
1 = The system switches to high frequency when the communication processor module is active.

PRQEN—Power Management Request Enable

This bit specifies whether or not the general system clock returns to a high frequency when a pending interrupt from the interrupt controller or POW bit in the machine state register is clear (normal mode). This bit is cleared by power-on or hard reset.

0 = The system remains in low frequency even if there is a pending interrupt from the interrupt controller or power management bit in the machine state register is cleared (normal mode).
1 = The system switches to high frequency when there is a pending interrupt from the interrupt controller or power management bit in the machine state register is cleared.

Bits 11–12 and 15–16—Reserved

These bits are reserved and should be set to 0.

EBDF—External Bus Division Factor

This field defines the frequency division factor between GCLKx and GCLKx_50. CLKOUT is similar to GCLK2_50. The GCLKx_50 is used by the bus interface and memory controller to interface with an external system. This field is initialized during hard reset using the hard reset configuration word in **Section 4.3.1.1 Hard Reset Configuration Word**.

00 = CLKOUT is GCLK2 divided by 1.
01 = CLKOUT is GCLK2 divided by 2.
1x = Reserved.

DFSYNC—Division Factor for the SYNCCLK

This field sets the VCOOUT frequency division factor for the SYNCCLK signal. Changing the value of this field does not result in a loss-of-lock condition. This field is cleared by a power-on or hard reset.

00 = Divide by 1 (normal operation).
01 = Divide by 4.
10 = Divide by 16.
11 = Divide by 64.

DFBRG—Division Factor of the BRGCLK

This field sets the VCOOUT frequency division factor for the BRGCLK signal. Changing the value of this field does not result in a loss-of-lock condition. This field is cleared by a power-on or hard reset.

00 = Divide by 1 (normal operation).
01 = Divide by 4.
10 = Divide by 16.
11 = Divide by 64.

DFNL—Division Factor Low Frequency

This field sets the VCOOUT frequency division factor for general system clocks to be used in low-power mode. In low-power mode, the MPC823 automatically switches to the DFNL frequency. To select the DFNL frequency, load this field with the divide value and set the CSRC bit. A loss-of-lock condition will not occur when you change the value of this field. This field is cleared by a power-on or hard reset.

000 = Divide by 2.
001 = Divide by 4.
010 = Divide by 8.
011 = Divide by 16.
100 = Divide by 32.
101 = Divide by 64.
110 = Reserved.
111 = Divide by 256.

DFNH—Division Factor High Frequency

This field sets the VCOOUT frequency division factor for general system clocks to be used in normal mode. In normal mode, the MPC823 automatically switches to the DFNH frequency. To select the DFNH frequency, load this field with the divide value and clear the CSRC bit. A loss-of-lock condition will not occur when you change the value of this field. This field is cleared by a power-on or hard reset.

000 = Divide by 1.
001 = Divide by 2.
010 = Divide by 4.
011 = Divide by 8.
100 = Divide by 16.
101 = Divide by 32.
110 = Divide by 64.
111 = Reserved.

DFLCD—Division Factor of LCDCLK

This field sets the VCOOUT frequency division factor for the LCDCLK signal. The total division factor of DFALCD and this field should not exceed 64, as defined in **Section 5.3.4.4 The LCD Clocks**. A loss-of-lock condition does not occur when you change the value of this field. This field is cleared by a power-on or hard reset.

000 = Divide by 1.
001 = Divide by 2.
010 = Divide by 4.
011 = Divide by 8.
100 = Divide by 16.
101 = Divide by 32.
110 = Divide by 64.
111 = Reserved.

DFALCD—Division Factor of LCDCLK50

This field sets the LCDCLK frequency division factor for the LCDCLK50 signal. The LCDCLK50 signal is input to the LCD panel. The total division factor of DFLCD and this field should not exceed 64, as defined in **Section 5.3.4.4 The LCD Clocks**. Changing the value of this field does not result in a loss-of-lock condition. This field is cleared by a power-on or hard reset.

00 = Divide by 1.
01 = Divide by 3.
10 = Divide by 5.
11 = Divide by 7.

5.2.2 PLL, Low-Power, and Reset Control Register

The 32-bit system PLL, low-power, and reset control register (PLPRCR) is powered by a keep-alive power supply and is used to control the system frequency and low-power mode operation.

PLPRCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	MF												RESERVED			
HRESET	—												0			
POR	*												0			
R/W	R/W												RW			
ADDR	(IMMR & 0xFFFF0000) + 0x284															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	SPLSS	TEXPS	RES	TMIST	RES	CSRC	LPM		CSR	LOLRE	FIOPD	RESERVED				
HRESET	—	1	0	0	0	0	0		—	—	—	0				
POR	0	1	0	0	0	0	0		0	0	0	0				
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W				
ADDR	(IMMR & 0xFFFF0000) + 0x286															

NOTE: HRESET is hard reset and POR is power-on reset.
— = Undefined.
* Depends on the combination of MODCK1and MODCK2. See Table 5-2 for more information.

MF—Multiplication Factor

The output of the voltage control oscillator (VCO) frequency is divided to generate the feedback signal that goes to the phase comparator. This field controls the value of the divider in the SPLL feedback loop. The phase comparator determines the phase shift between the feedback signal and the reference clock. This difference results in an increase or decrease of the VCO output frequency.

The MF field can be read and written at any time. Changing the MF field causes the SPLL to lose its lock. All clocks are disabled until the SPLL reaches lock condition. The normal reset value for the DFNH bits is 0x0 (divide-by-one). When the SPLL is operating in one-to-one mode, the MF field is set to 0. See Table 5-2 for details.

Bits 12–15—Reserved

These bits are reserved and should be set to 0.

SPLSS—System PLL Lock Status Sticky

This bit is not affected by hard reset. An out-of-lock indication sets the SPLSS bit and it remains set until the software clears it. At power-on reset, the state of the SPLSS bit is zero. Write a 1 to clear this bit (writing a zero has no effect). Notice that a loss-of lock caused by a change in the MF field or the processor entering deep-sleep mode or power-down mode does not affect the SPLSS bit. Only a loss-of-lock caused by the following conditions will set this bit.

0 = SPLL remains locked.
1 = SPLL has gone out of lock at least once since the bit was cleared or at the last power-on reset.

TEXPS—Timer Expired Status

This internal status bit is set when the periodic timer expires, the real-time clock alarm sets, the timebase clock alarm sets, the decrementer interrupt occurs, or the system resets. The TEXP pin reflects the value of the TEXPS bit, so you have to read the pin to find out the status of the bit. You can clear this bit by writing a 1 (writing a zero has no effect). When TEXPS is set, the TEXP external signal is asserted and when it is reset, the TEXP external signal is negated.

0 = TEXP is negated.
1 = TEXP is asserted.

Bits 18, 20, and 27–31—Reserved

These bits are reserved and should be set to 0.

TMIST—Timers Interrupt Status

This bit is cleared at reset and is set when a real-time clock, periodic interrupt timer, timebase, or decrementer interrupt occurs. You can clear this bit by writing a 1 (writing a zero has no effect). When the TMIST bit is set, the system clock switches to high frequency, as defined by the DFNH field. The system clock frequency stays high if the CSRC bit is set and there is no reason to switch to normal low mode.

0 = No timer interrupt is detected.
1 = A timer interrupt is detected.

CSRC—Clock Source

This bit specifies whether the DFNH or DFNL field generates the general system clock. This bit is cleared by a hard reset.

0 = The general system clock is generated by the DFNH field.
1 = The general system clock is generated by the DFNL field.

LPM—Low-Power Modes

This bit, in conjunction with the TEXPS and CSRC bits, specifies the operating mode of the core. There are seven possible modes. In the normal mode, you can write a non-zero value to this field. In the other modes, only a reset or interrupt (that is not from the interrupt controller) can clear this field.

00 = Normal high/normal low mode.
01 = Doze high/doze low mode.
10 = Sleep mode.
11 = Deep sleep/power-down mode.

CSR—Checkstop Reset Enable

This bit enables an automatic reset when the processor enters checkstop mode. If the processor enters debug mode at reset, then reset is not generated automatically. Refer to **Section 20.6.3.2 Debug Enable Register** for more information.

CSR BIT	CHSTPE BIT IN DER	CHECKSTOP MODE	RESULT
0	0	No	—
0	0	Yes	—
0	1	No	—
0	1	Yes	Enter debug mode
1	0	No	—
1	0	Yes	Automatic reset
1	1	No	—
1	1	Yes	Enter debug mode

LOLRE—Loss-of-Lock Reset Enable

This bit enables hard reset generation when a loss-of-lock indication occurs, but not as a result of altering the MF field or the processor entering deep-sleep or power-down mode.

0 = A hard reset is not generated when a loss-of-lock is indicated.
1 = A hard reset is generated when a loss-of-lock is indicated.

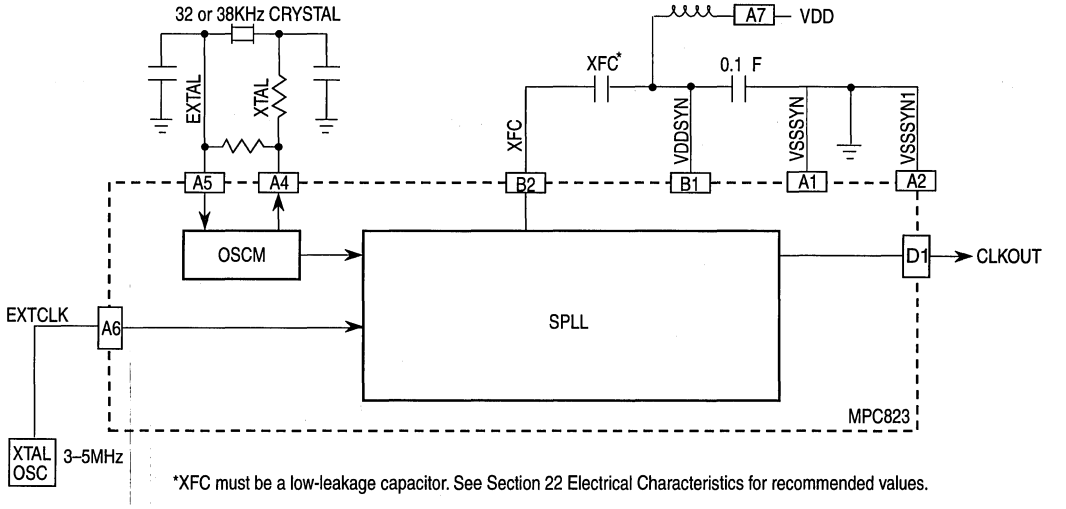
FIOPD—Force I/O Pull Down

This bit indicates when the address and data external pins are driven by an internal pull-down device in sleep and deep-sleep mode.

0 = No pull-down on the address and data bus.
1 = Address and data bus is driven low in sleep and deep-sleep mode.

5.3 THE CLOCK MODULE

The input frequency source for the phase-locked loop can either be a low frequency crystal (32kHz or 38.4kHz) or a high frequency oscillator (3-5MHz) that the phase-locked loop multiplies to produce the system clock. If you use a low frequency crystal circuitry, the on-chip main clock oscillator requires an external parallel resonant crystal, two capacitors, and two resistors, as shown in Figure 5-2. When you design and implement this circuitry, follow normal high-frequency design rules for placement and layout.



*XFC must be a low-leakage capacitor. See Section 22 Electrical Characteristics for recommended values.

Figure 5-2. Crystal Oscillator

If you use a high-frequency oscillator, its output can be directly input to the EXTCLK pin. Even though crystals are typically much cheaper, oscillators have more stability because they are less affected by trace length, component quality, printed circuit board layout and characteristics.

Note: In your design, you should always consider the oscillation startup stablization time of the crystal. During this stabilization time, no internal or external clocks are generated by the MPC823, which may cause higher than normal static current during the short period of stabilization. Assuming that the crystal circuit is powered by keep-alive power, then once you system has had first time power-up, allowance for stabilization is not necessary.

The MPC823 clock module consists of the main crystal oscillator, the SPLL, the low-power divider, the clock generator/driver blocks, and the clock module/system low-power control block. The clock module and system low power control block receive control bits from the system clock control register (SCCR), PLL low-power and reset control register (PLPRCR), and reset status register (RSR). Figure 5-3 illustrates the clock module.

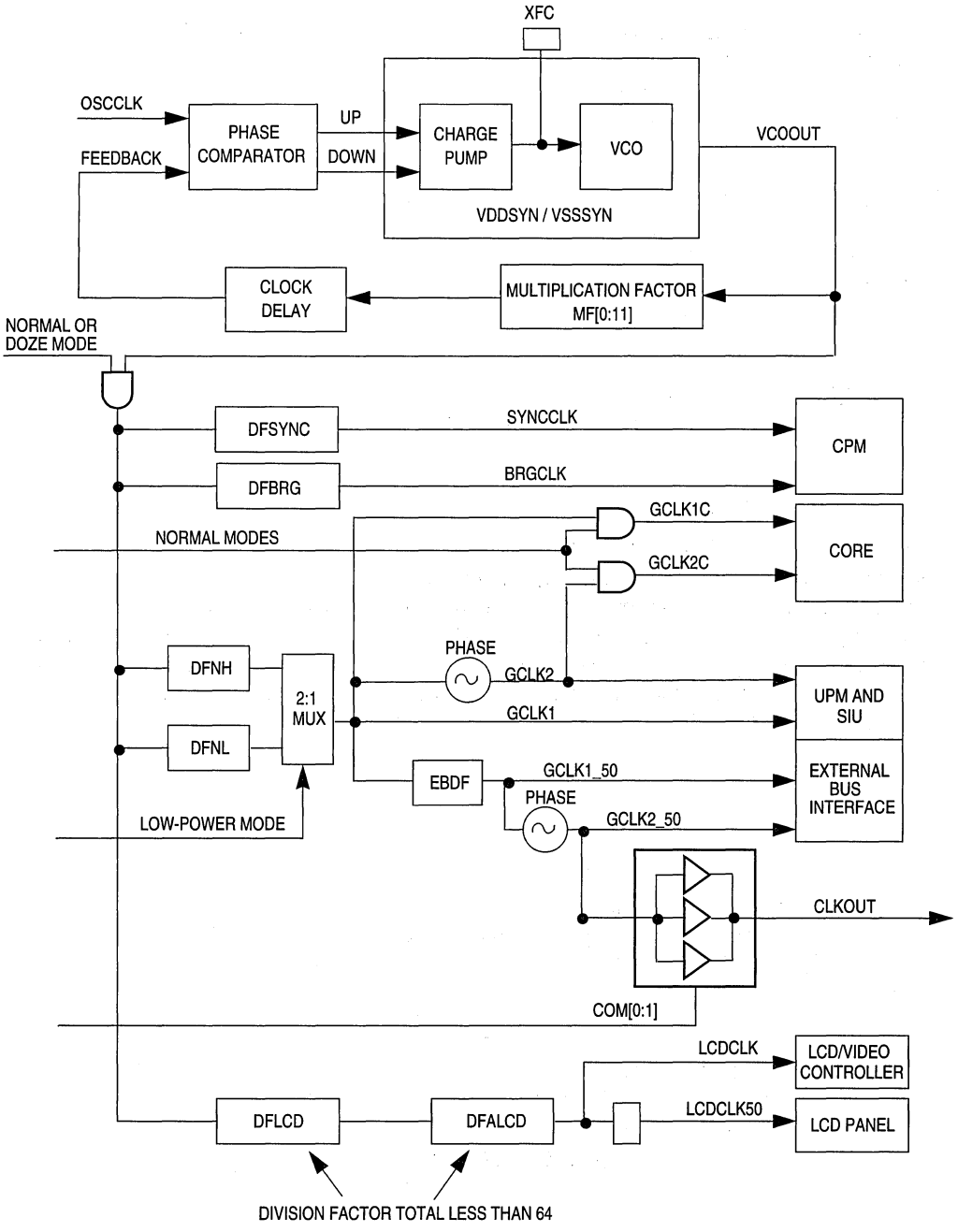


Figure 5-3. Clock Module Diagram

5.3.1 On-Chip Oscillators and External Clock Input

The main clock oscillator (OSCM) uses either a 3-5MHz source at EXTCLK (4MHz mode) oscillator or a 30-50kHz (32kHz mode) crystal between EXTAL and XTAL to generate the SPLL reference clock (OSSCLK). The external clock input EXTCLK is generated from an external source. In one-to-one mode, the input clock frequency must be at least 15MHz. Otherwise, it can be between 3 and 5MHz. See Figure 5-1 for details.

For normal operation, at least one clock source should be active, but it is possible to configure both clock sources to be active. When both clock sources are active, the EXTCLK pin provides the OSCCLK signal for the SPLL. The EXTAL and XTAL pins provide the input frequency for the OSCM that generates the PITRTCLK signal. The input of an unused timing reference should be grounded.

Note: Under any condition, the voltage on the MODCK1 and MODCK2 pins should be less than or equal to the power supply voltage VDDH applied to the part.

5.3.2 System PLL

The main purpose of the SPLL is to generate a stable reference frequency by multiplying the frequency and eliminating the clock skew. The SPLL allows the processor to operate at a high internal clock frequency using a low frequency clock input, which gives you two advantages. Lower frequency clock input reduces the overall electromagnetic interference generated by the system. Also, oscillating at different frequencies reduces the cost because you will not have to add more oscillators to your system. The MPC823 SPLL block diagram is illustrated in Figure 5-4.

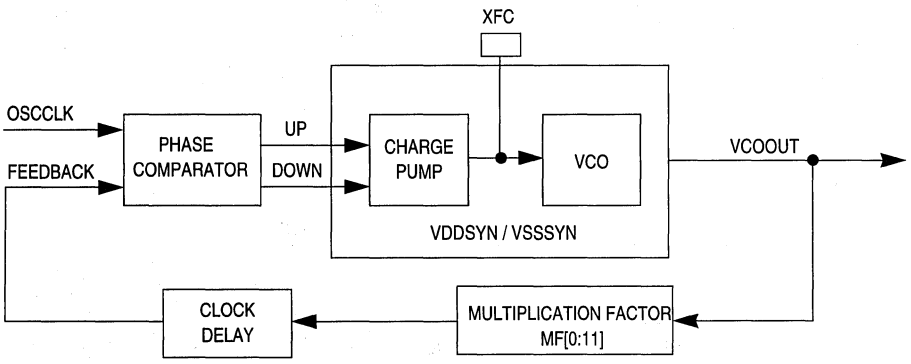


Figure 5-4. SPLL Block Diagram

The OSCCLK signal goes to the phase comparator that controls the direction in which the charge pump drives the voltage across the external filter capacitor (XFC). Direction is based on whether the feedback signal phase lags or leads the reference signal. The output of the charge pump drives the VCO whose output frequency at VCOOUT is divided down and fed back to the phase comparator to be compared with the reference frequency (OSCCLK signal). The multiplication factor should be between 1 and 4,096. Also, when the SPLL is operating in one-to-one mode, the multiplication factor is 1 (the MF field equals 0).

At the initial system power-up after keep-alive power is lost, external logic should assert the $\overline{\text{PORESET}}$ pin for 3µs after a valid level is reached on the keep-alive power supply. When power-on reset is asserted, the MF field is set as shown in Table 5-1 and the DFNH and DFNL fields are both set to 0. This MF field value then programs the SPLL to generate the approximate default system frequency of 16.7MHz when a 32kHz input frequency is used and 20MHz when a 4MHz input frequency is used.

Table 5-1. Power-On Reset Clock Configuration

MODCK [1:2]	DEFAULT MF + 1 AT POWER-ON RESET	SPLL OPTIONS
00	513	Normal operation, SPLL enabled. Main timing reference is $OSCM_{freq}$ = 32 kHz.
01	5	Normal operation, SPLL enabled. Main timing reference is $OSCM_{freq}$ = 4 MHz.
11	5	Normal operation, SPLL enabled. Main timing reference is $EXTCLK_{freq}$= 4 MHz.
10	1	Normal operation, SPLL enabled. One-to-one Mode Maximum $CLKOUT_{freq}$ = $EXTCLK_{freq}$

5.3.2.1 SPLL STABILITY. The SPLL can multiply the input frequency by any integer that is between 1 and 4,096. The multiplication factor can be changed by modifying the value of the MF field in the PLPRCR. Even though any multiplication factor between 1 and 4,096 can be programmed, the resulting VCO output frequency must be within the range specified in **Section 22 Electrical Characteristics**. The MF field in the PLPRCR is set to a predetermined value when a power-on reset occurs. The multiplication factor is the most important parameter and as it goes higher it has a greater effect on the stability of the SPLL. There are three factors that define SPLL stability—phase skew, phase jitter, and frequency jitter.

The phase skew is the time difference between the falling edges of the EXTAL and CLKOUT pins for a capacitive load on CLKOUT over the entire process, temperature ranges, and voltage ranges. For input frequencies greater than 15MHz and MF≤2, this skew is between -0.9ns and +0.9ns. Otherwise, this skew is not guaranteed. However, for MF<10 and input frequencies greater than 10MHz, the skew is between -2.3ns and +2.3ns.

The phase jitter is a variation in the skew that occurs between the falling edges of the EXTAL and CLKOUT pins for a specific temperature, voltage, input frequency, MF, and capacitive load on the CLKOUT pin. These variations are a result of the PLL locking mechanism. For input frequencies greater than 15MHz and MF≤2, this jitter is less than ±0.6ns. Otherwise, this jitter is not guaranteed. However, for MF<10 and input frequencies greater than 10MHz, this jitter is less than ±2ns.

The frequency jitter is defined as the frequency variation of the CLKOUT pin. For small multiplication factors (MF<10), this jitter is smaller than 0.5%. For mid-range multiplication factors (10<MF<500), this jitter is between 0.5% and -2%. For large multiplication factors (MF>500), the frequency jitter is 2–3%. The maximum input frequency jitter on the EXTAL pin is 0.5%. If the rate of change of the frequency at the EXTAL pin is slow (it does not jump between the minimum and maximum values in one cycle), the maximum jitter can be 2%.

5.3.3 The Low-Power Clock Divider

The output of the SPLL is sent to a low-power divider that generates other clocks for normal operation, but also has the ability to divide the output frequency of the VCO before it generates the SYNCCLK, LCDCLK, LCDCLK50, BRGCLK, and GCLKx (which is sent to the rest of the MPC823). GCLKxC is the system timing reference for the core, instruction and data caches, and memory management unit. GCLKx is the system timing reference for the other modules. GCLKx_50 operates at a frequency that is half the GCLKx frequency. The frequency ratio between GCLKx and GCLKx_50 is determined by the EBDF bit in the SCCR.

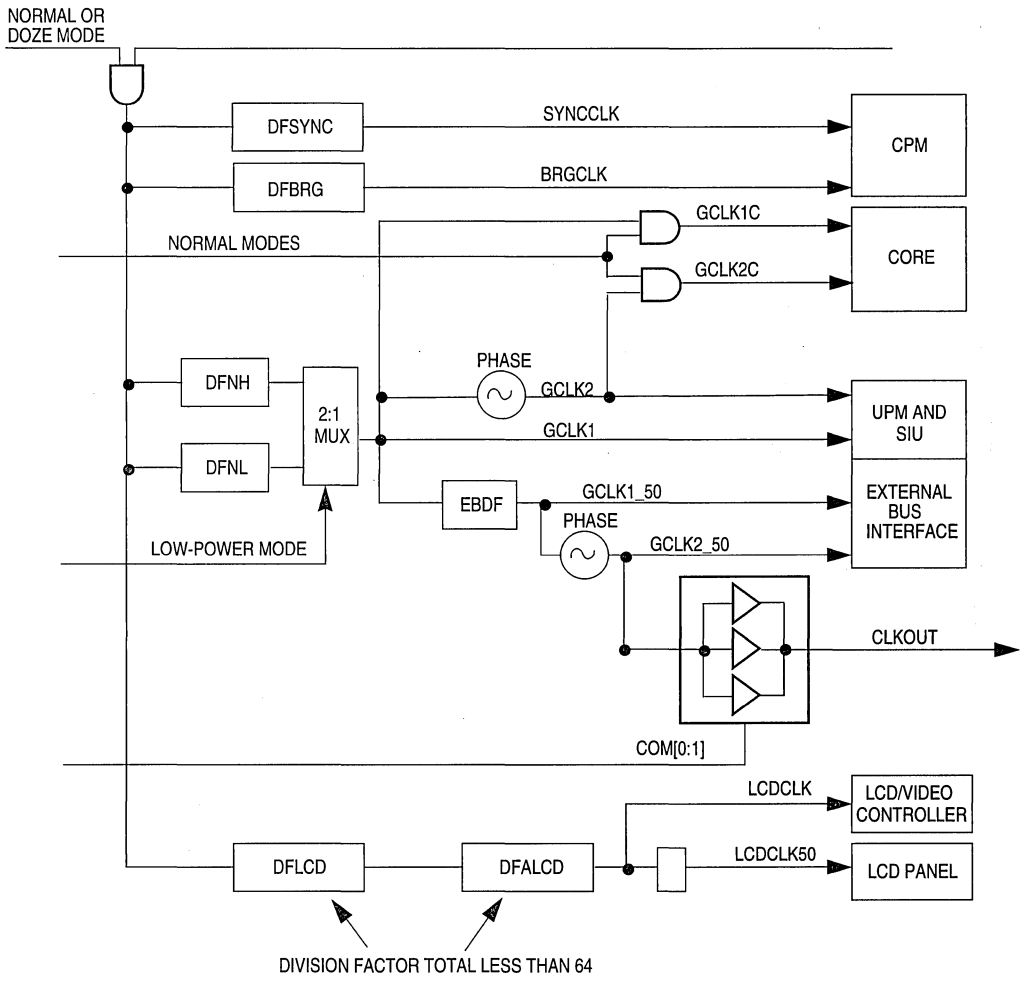


Figure 5-5. Clock Dividers

The low-power dividers allow you to reduce and restore the operating frequencies of different sections of the MPC823 without losing the SPLL lock. Using the low-power dividers, you can still obtain full chip operation, but at a lower frequency. This is called normal low mode. You can switch to normal low mode or set the speed at any time and the changes you make will occur immediately. The low-power dividers are controlled by the SCCR and its default division factor is divide by one at reset. For a 40MHz system, SYNCCLK, LCDCLK, LCDCLK50, BRGCLK, and GCLKx are all 40MHz at reset.

5.3.4 Internal Clock Signals

The internal logic of the MPC823 uses the following internal clock signals:

- General system clocks
- Baud rate generator clock
- Synchronization clocks
- LCD clocks

The MPC823 also generates an external clock signal called CLKOUT. All internal clock signals originate from the same source, so they are all synchronized to VCOOUT.

5.3.4.1 THE GENERAL SYSTEM CLOCKS. The general system clocks—GCLK1C, GCLK2C, GCLK1, GCLK2, GCLK1_50, and GCLK2_50—are the basic clocks supplied to all modules of the MPC823. GCLKxC is supplied to the core, data and instruction caches, and memory management unit. It is not active when the core is in sleep or power-down mode. GCLKx is supplied to the system interface unit, clock module, memory controller, and most of the other blocks in the communication processor module.

The timing relationship between the general system clock signals that are shown in Figure 5-1 is illustrated in Figure 5-6. CLKOUT, the only externally visible clock, is derived from the GCLK2_50 signal.

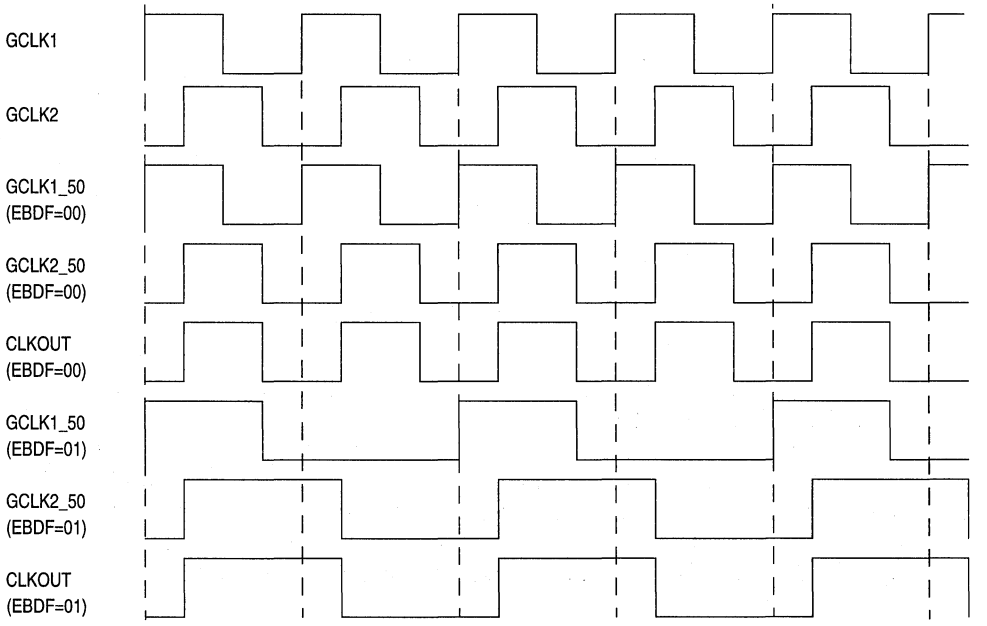


Figure 5-6. MPC823 Clocks Timing Diagram

Notice that GCLK1_50, GCLK2_50, and CLKOUT can have a lower frequency than GCLK1 and GCLK2. This allows the external bus to operate at lower frequencies as controlled by the EBDF bit in the SCCR. GCLK2_50 always rises simultaneously with GCLK2. GCLK1_50 rises simultaneously with GCLK1, but when the MPC823 is not in normal low mode, the falling edge of GCLK1_50 occurs in the middle of the high phase of GCLK2_50 and EBDF determines the division factor between GCLKx and GCLKx_50. See Figure 5-9 for more information.

The general system clock defaults to VCOOUT frequency at system reset. In normal low mode, the frequency of the general system clock can be dynamically switched using the SCCR.

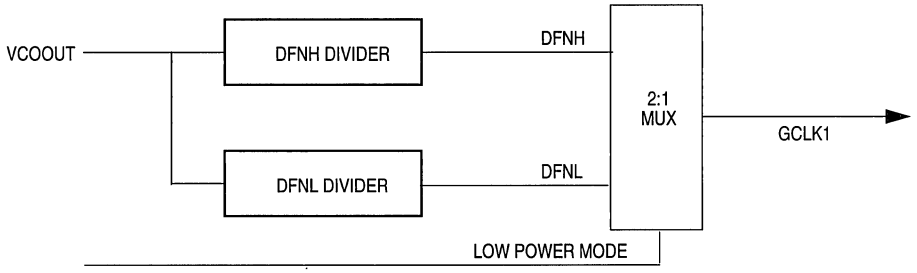


Figure 5-7. Selecting the General System Clock

You can switch the general system clock division factor between two different values (DFNH and DFNL). The high frequency is generated by using the DFNH field in the SCCR and it is used in normal high and doze high mode. The low frequency is generated using the DFNL field in the SCCR and it is used in normal low and doze low mode. Conventionally, to conserve power low frequency is slower than high frequency.

In some applications, a high frequency is needed to perform critical tasks. For example, interrupt routines need to be run at a high frequency, but the rest of the application can run at a low frequency to conserve power. The MPC823 can automatically switch between low and high frequency operation when one of the following conditions exist:

- A pending interrupt from the interrupt controller occurs. This option is maskable by the PRQEN bit in the SCCR.
- The POW bit of the machine status register is clear. This option is maskable by the PRQEN bit in the SCCR.
- The RISC microcontroller in the communication processor module has a pending request or is currently executing a routine. This option is maskable by the CRQEN bit in the SCCR.

When none of these conditions exist and the CSRC bit of the PLPRCR is set, the general system clock automatically switches back to low frequency. When the general system clock is divided, its duty-cycle is modified. One phase remains the same while the other stretches out. GCLKx and CLKOUT no longer have a 50% duty cycle when the division factor is greater than 1, as shown in Figure 5-8 and Figure 5-9.

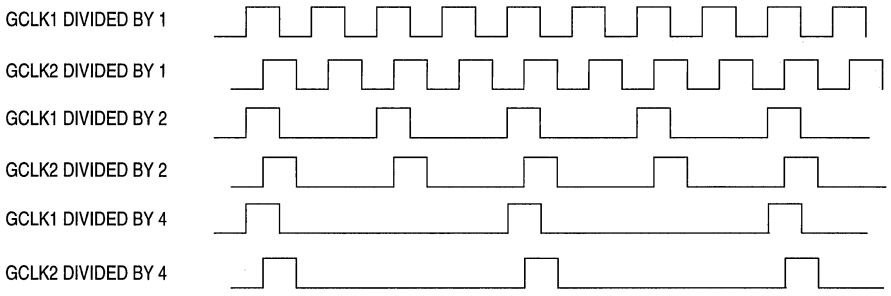


Figure 5-8. Divided System Clocks Timing Diagram

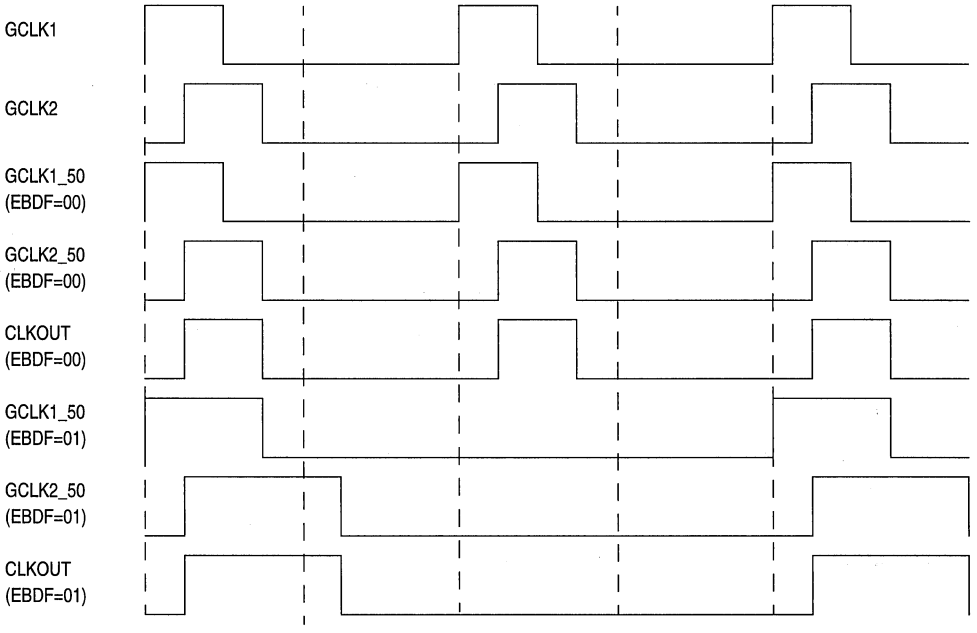


Figure 5-9. MPC823 Clocks For Division Factor 2

The frequency for the GCLKx system clock is:

$$GCLKx_{freq} = \frac{VCOOUT_{freq}}{(2^{DFNH})or(2^{DFNL+1})}$$

The frequency for the GCLKx_50 system clock is:

$$GCLKx\_50_{freq} = \frac{VCOOUT_{freq}}{(2^{DFNH})or(2^{DFNL+1})} \times \frac{1}{EBDF+1}$$

CLKOUT is derived from GCLK2_50. It defaults to VCOOUT, which is the user-defined system frequency (25 or 50MHz). CLKOUT can drive at full-strength, half-strength, or it can be disabled. The strength of the drive is controlled in the system clock and reset control register. Disabling or decreasing the strength of CLKOUT reduces power consumption, noise, and electromagnetic interference on the printed circuit board. When the SPLL is acquiring lock, the CLKOUT signal is disabled and remains in a low state.

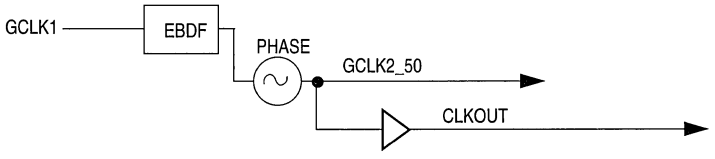


Figure 5-10. CLKOUT Divider

5.3.4.2 THE BAUD RATE GENERATOR CLOCK. The baud rate generator clock (BRGCLK) is used by the four baud rate generators of the communication processor module and by the memory controller refresh counter. It defaults to VCOOUT, which is the user-defined system frequency (25 or 50MHz). The baud rate generator clock allows the baud rate generators to continue operating at a fixed frequency, even when the rest of the MPC823 is operating at a reduced frequency. Refer to **Section 16.8 The Baud Rate Generators** for more information about using the baud rate generator clock to save power.

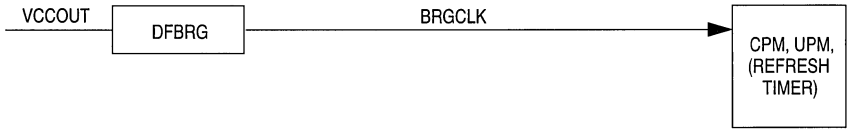


Figure 5-11. BRGCLK Divider

The baud rate generator clock frequency is:

$$BRGCLK_{freq} = \frac{VCOOUT_{freq}}{(2^{2 \times DFBRG})}$$

5.3.4.3 THE SYNCHRONIZATION CLOCKS. The synchronization clock signal (SYNCCLK) is used by the serial synchronization circuitry in the serial ports of the communication processor module that includes the serial interface, serial communication controller, and serial management controllers. SYNCCLK defaults to VCOOUT, which is the user-defined system frequency (25 or 50MHz).

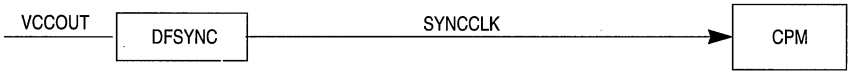


Figure 5-12. SYNCCLK Divider

SYNCCLK allows the serial interface, serial communication controller, and serial management controllers to continue operating at a fixed frequency, even when the rest of the MPC823 is operating at a reduced frequency. This allows you to maintain the serial synchronization circuitry at the preferred rate, while lowering the general system clock to the lowest possible rate. SYNCCLK must always have a frequency at least as high as the general system clock frequency and be at least two times the preferred serial clock rate. If the time-slot assigner in the serial interface is used, SYNCCLK is at least two and half times the preferred serial clock rate. Refer to **Section 16.7 The Serial Interface with Time-Slot Assigner** for more information on how to select an appropriate frequency for SYNCCLK. The synchronization clock frequency is:

$$SYNCCLK_{freq} = \frac{VCOOUT_{freq}}{(2^{2 \times DFSYNC})}$$

5.3.4.4 THE LCD CLOCKS. The LCD clocks—LCDCLK and LCDCLK50—are used by the LCD controller circuitry to transfer the frame data to pixel format data. LCDCLK defaults to VCOOUT, which is the user-defined system frequency (25 or 50MHz).When the PON bit in the LCCR is set, the ratio between the system clock frequency value and the LCD clock frequency value should be an integer value. The LCD clock frequency is the system frequency divided by two serial dividers. LCDCLK50 is a 50% duty-cycle clock at the same frequency as LCDCLK that is used as a clock output to the LCD panel. DFLCD and DFALCD should be set so that the total LCD clock division factor never exceeds 64.

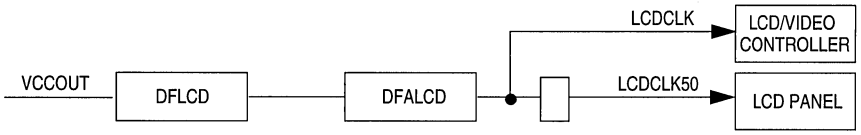


Figure 5-13. LCDCLK Divider

The LCDCLK and LCDCLK50 frequency is:

$$LCDCLK_{freq} = LCDCLK50_{freq} = \frac{VCOOUT_{freq}}{(2^{DFLCD}) \times (2 \times DFALCD + 1)}$$

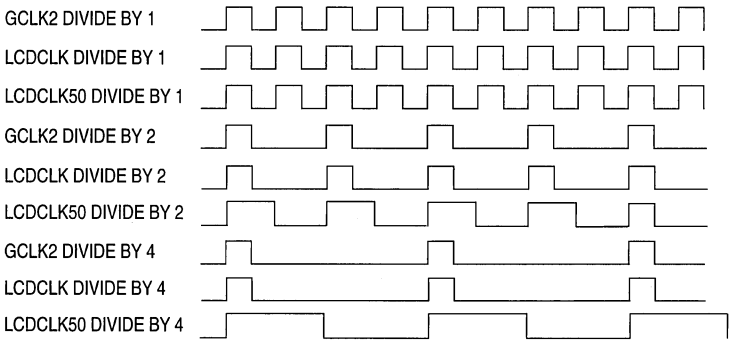


Figure 5-14. LCD Clock Timing Diagram

5.3.5 Clock Configuration

You can configure the clock of the MPC823 using the MODCK1 and MODCK2 pins. The SPLL has several power and ground pins (VDDSYN, VSSSYN, VSSSYN1, and XFC) that must be properly terminated for stability and CLKOUT integrity.

5.3.5.1 MODE CLOCK PINS. The MODCK1 and MODCK2 pins are used to determine clock source and configuration options. These pins are sampled on the rising edge of the $\overline{\text{PORESET}}$ pin. MODCK1 specifies the input source to the SPLL and, combined with MODCK2, specifies the multiplication factor at reset. If you do not want the PITRTCLK, TMBCLK, and multiplication factor to change during power-down mode, do not sample the MODCK1 and MODCK2 pins during wake-up. Thus, the $\overline{\text{PORESET}}$ pin should remain negated while the $\overline{\text{HRESET}}$ pin is asserted during this wake-up sequence.

Table 5-2. Reset Clock Source Configuration

MODCK [1:2]	POWER-ON RESET (POR)	DEFAULT MF + 1 AT POR	PITRTCLK DIVISION DEFAULTS AT POR	TMBCLK DIVISION DEFAULTS AT POR	SPLL OPTIONS
00	0	513	4	4	Normal operation, SPLL enabled. Main timing reference is $OSCM_{freq}$ = 32 kHz.
01	0	5	512	4	Normal operation, SPLL enabled. Main timing reference is $OSCM_{freq}$ = 4 MHz.
11	0	5	512	4	Normal operation, SPLL enabled. Main timing reference is $EXTCLK_{freq}$ = 4 MHz.
10	0	1	512	16	Normal operation, SPLL enabled. One-to-one Mode, Maximum $CLKOUT_{freq}$ = $EXTCLK_{freq}$
—	1	—	—	—	The configuration remains unchanged.

MODCK1 selects the clock source to use (EXTCLK or main oscillator) and MODCK2 selects the reference frequency (4MHz, 32kHz, or EXTCLK). If, during the power-on reset sequence, the MODCK1 pin is zero, then the oscillator's 4MHz or 32kHz clock is the selected input. If MODCK1 is one, then the EXTCLK pin is the input source to the SPLL. The system clock frequency (GCLK1) can be divided as defined by the DFNH and DFNL fields in the SCCR. The maximum system clock frequency occurs when the DFNH field is equal to 00. If MODCK2 is zero during the power-on reset sequence, the input frequency is either 32kHz or EXTCLK (one-to-one mode). If the MODCK2 pin is one, then a 4MHz clock is the input source to the SPLL.

If the EXTCLK pin is the main timing reference and the OSCM is the timing reference to the real-time clock and periodic interrupt timer, then the frequency of the OSCM should be 32.768 or 38.4kHz. Depending on how the TBS bit is set in the SCCR, the timebase clock is either the SPLL clock frequency or GCLK2. The RTSEL field in the SCCR specifies the source (EXTCLK or OSCM) of the periodic interrupt timer and real-time clock. The RTDIV field in the SCCR specifies the division factor (÷ by 4 or ÷ by 512).

Table 5-3. TMBCLK Dividers

TBS BIT IN SCCR	MODCK1 AT RESET	MF + 1	CLOCK SOURCE	TMBCLK DIVISION
1	—	—	GCLK2	16
0	0	—	OSCM	4
0	1	1, 2	EXTAL	16
0	1	> 2	EXTAL	4

5.3.5.2 THE SYSTEM PHASE-LOCKED LOOP PINS. The internal frequency of the MPC823 and the output of the CLKOUT pin depends on the quality of the crystal circuit and the MF bit in the PLPRCR. The SPLL contains the following dedicated pins that are isolated from common power and ground.

- VDDSYN—The power supply pin for the analog SPLL circuitry. You should provide a well-regulated voltage to this pin via an extremely low impedance path to the VDD power rail. The SPLL power plane must be isolated from the VDD power plane with an LC filter, which gives you the best performance. A 0.1µF bypass capacitor must connect VDDSYN and VSSSYN and be as close to the chip as possible. To reduce the noise from the open EXTAL pin, pull it to ground when you are not using it.
- VSSSYN—A ground reference pin for the analog SPLL circuitry. You should provide a low impedance path from VSSSYN to the ground plane.
- VSSSYN1—A ground reference pin for the analog SPLL circuitry. You should provide a low impedance path from VSSSYN1 to the ground plane.
- XFC—The external filter capacitor pin that connects to the off-chip capacitor for the SPLL filter. One terminal of the capacitor is connected to XFC while the other terminal is connected to the VDDSYN pin. For proper SPLL operation, you should supply a minimum parallel parasitic resistance value of 30MΩ. The value of the XFC capacitor is based on the value of the MF field in the PLPRCR.

Table 5-4. XFC Capacitor Values Based on the MF Field

MF RANGE	MINIMUM CAPACITANCE	MAXIMUM CAPACITANCE	UNIT
MF ≤ 4	XFC = (MF x 425) - 125	XFC = (MF x 590) - 175	pF
MF > 4	XFC = MF x 520	XFC = MF x 920	pF

5.4 POWER CONTROL

To preserve the life of your battery, the MPC823 provides low-power modes that limit the operation to essential modules. In addition to normal high mode, the MPC823 supports normal low, doze high, doze low, sleep, deep-sleep, and power-down modes. When the CPM is idle, it uses its own power-saving mechanism to shut down automatically. There are a variety of modules connected to separate power rails that allow you to achieve the best performance using the least amount of power.

5.4.1 Power Rails

There are different power planes (called power rails) within the MPC823 that can be connected to different power sources. Your system implementation defines the power source for each of these different power rails. Figure 5-15 shows you which module is connected to each power rail.

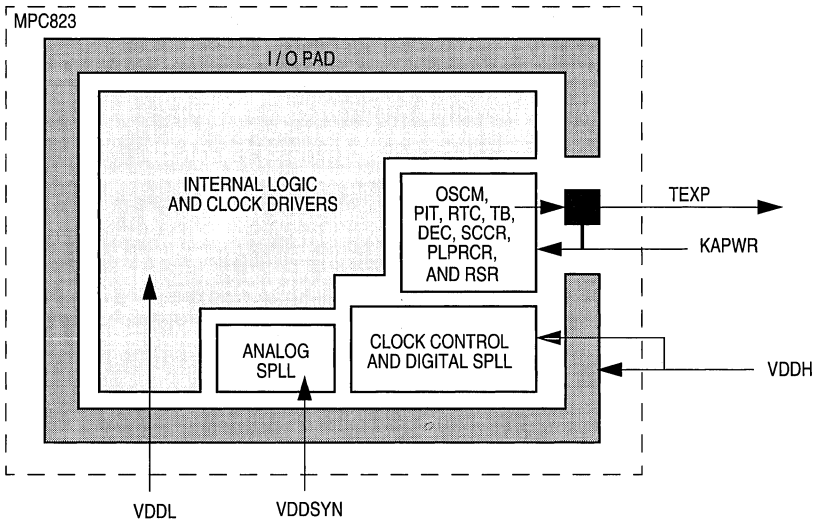


Figure 5-15. MPC823 Power Rails and TEXP Status

The I/O buffers, logic, and clock control are fed by a 3.3V (±10%) power supply. The internal logic can be fed by a 3.3V source or, to conserve power, a 2.2V source. To improve CLKOUT stability, the SPLL requires a separate 3.3V source (VDDSYN). The OSCM, timebase, decrementer, periodic interrupt timer, real-time clock, SCCR, PLPRCR, and RSR are all connected to the keep-alive power (KAPWR) rail. This power rail architecture allows you to remove the power at the VDDH/VDDL/VDDSYN pins during sleep, deep-sleep, or power-down mode. The external power supply unit can use the TEXP pin, which is fed by the KAPWR rail, to switch between power sources.

All other circuits are powered by the normal supply pins—VDDH and VDDL—and VSS. VDDH feeds the I/O buffers and logic and VDDL supplies the internal chip logic to reduce system power consumption. However, the power supply connected to VDDH should be larger or equal to the one connected to VDDL. The power supply for each block is listed in Table 5-5 and described in **Section 5.4 Power Control**.

Table 5-5. MPC823 Power Supply

BLOCK	VDDH	VDDL	VDDSYN	KAPWR
I/O Pad	X			
CLKOUT	X			
Digital SPLL	X			
Clock Control	X			X
Internal Logic		X		
Clock Drivers		X		
Analog SPLL			X	
OSCM				X
SCCR, PLPRCR, and RSR				X
RTC, PIT, TB, and DEC				X

The following are the power supply requirements of the MPC823:

- VDDH = VDDSYN = 3.3V ±10%
- VDDH ≥ VDDL ≥ 2.2V ±10%
- VDDH ≥ KAPWR ≥ VDDH – 0.4V for normal operation
- KAPWR ≥ 2.2V ±10% in power-down mode

5.4.2 Keep-Alive Power

When the MPC823 is in normal operation mode, the keep-alive power supply (KAPWR) is powered to the same voltage value as that of the I/O buffers and logic. Therefore, if the VDDL is 2.2V and the VDDH is 3.3V, then the KAPWR is 2.9V to 3.3V.

5.4.2.1 POWER SWITCHING EXAMPLE. An example of a switching scheme for an optimized low-power system is illustrated in Figure 5-16.

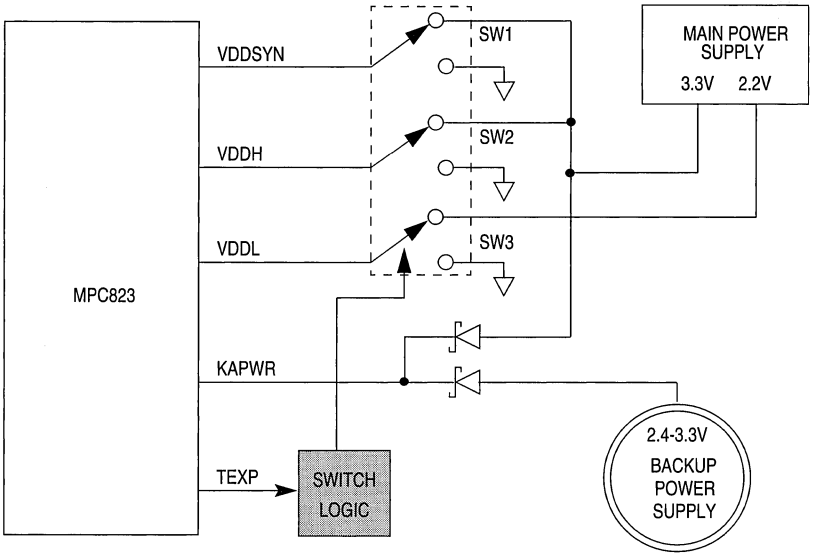


Figure 5-16. External Power Supply Scheme (2.2V Internal Voltage)

Switch 1 and 2 (SW1 and SW2) can be combined into a single switch if VDDSYN and VDDH are supplied by the same source. If VDDL is fed with 3.3V, SW2 and SW3 can be combined into one switch. The TEXP pin, if enabled, is asserted by the MPC823 when the real-time clock or timebase time value matches the value programmed in its associated alarm register or when the periodic interrupt timer or decrementer decrements their value to zero. The TEXP pin is negated when you write a 1 to the TEXPS bit in the PLPRCR.

If the voltage to the KAPWR rises too slow, the OSCM (supplied by KAPWR) will take longer to stabilize the OSCCLK. The maximum KAPWR rise time should be less than 1.7V/ms for a 32kHz input frequency.

5.4.2.2 REGISTER LOCK. The MPC823 registers that are powered by KAPWR can be write-protected using the associated key register shown in Table 5-6. When the MPC823 disconnects from the main power supply after it enters power-down mode, the value of these registers is automatically preserved. You may lose data if you do not enter power-down mode before disconnecting. To protect your data, lock the register by writing to the associated key register. Once you lock a register, writes are ignored and a machine check exception is generated.

Table 5-6. Key Registers

MNEMONIC	REGISTER NAME
TBSCRK	Timebase Status and Control Register Key
TBREFF0K	Timebase Reference Register 0 Key
TBREFF1K	Timebase Reference Register 1 Key
TBK	Timebase and Decrementer Register Key
RTCSCK	Real-Time Clock Status and Control Register Key
RTCK	Real-Time Clock Register Key
RTSECK	Real-Time Alarm Seconds Key
RTCALK	Real-Time Alarm Register Key
PISCRK	Periodic Interrupt Status and Control Register Key
PITCK	Periodic Interrupt Count Register Key
SCCRK	System Clock Control Key
PLPRCRK	PLL, Low Power and Reset Control Register Key
RSRK	Reset Status Register Key

NOTE: Refer to **Section 3 Memory Map** for each register's address.

Each of the registers in the keep-alive power region have a key register that can be in an open or locked state. At power-on reset, all key registers are open, except for the key real-time clock registers. Each key register has an associated address in the internal memory map, as shown in Table 3-1. If you write 0x55CCAA33 to any of the key registers, the associated register will be open and if you write any other value it will be locked. For example, writing a 0x55CCAA33 to the RTCK key register will allow you to access the RTC register.

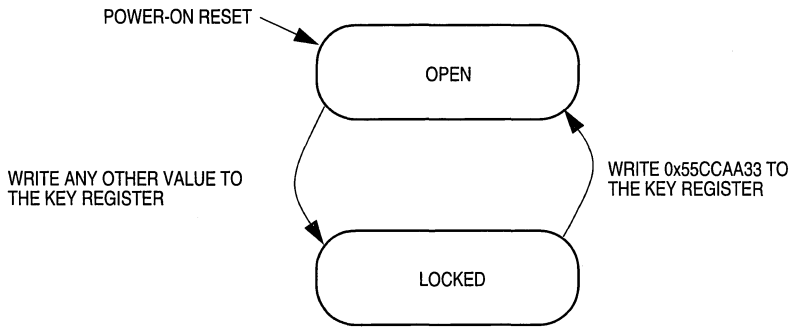


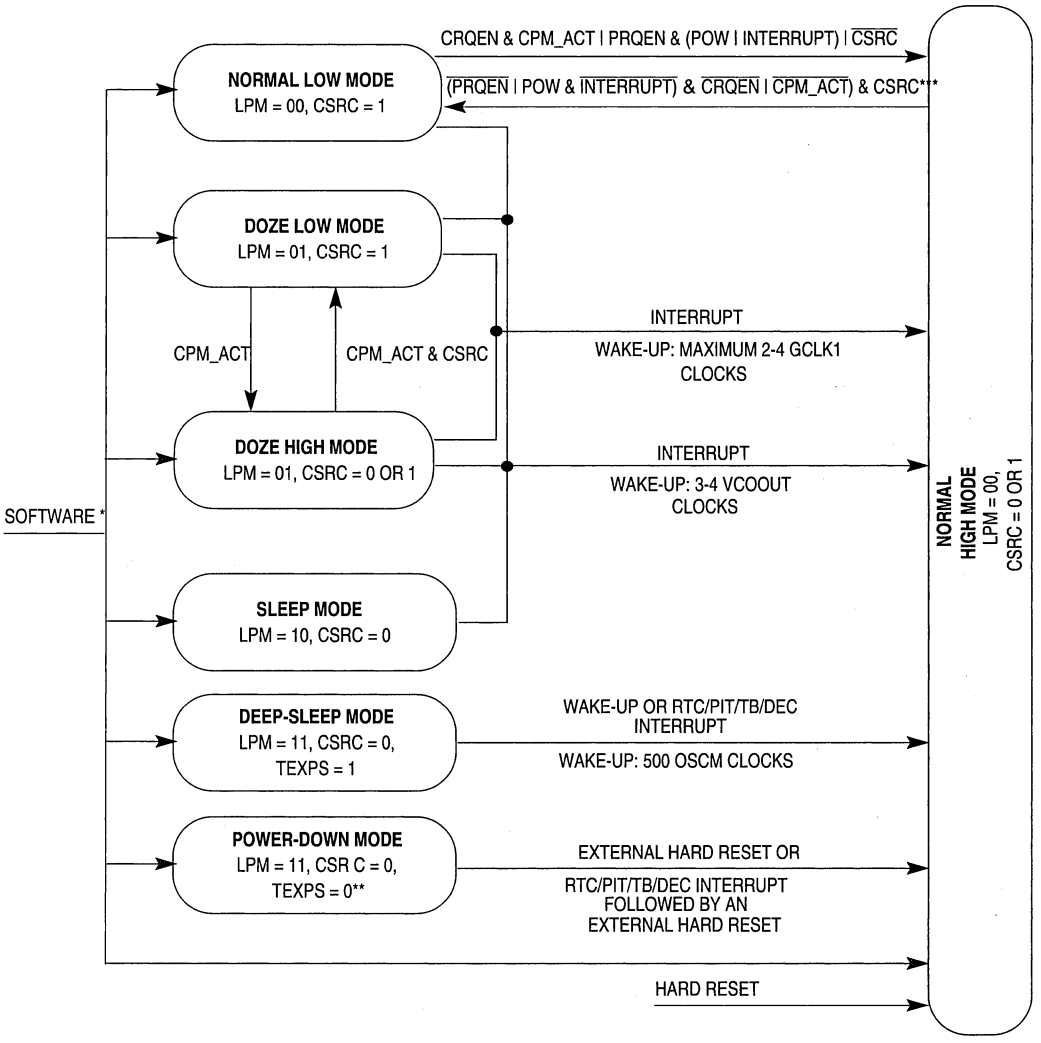
Figure 5-17. Register Lock Mechanism

5.5 LOW-POWER OPERATION

The low-power dividers can be used to take advantage of the MPC823's low-power capabilities. The low-power dividers allow you to dynamically adjust the operating frequencies for different modules of the MPC823 and yet still maintain the SPLL lock. In normal low mode, you can use the low-power dividers to maintain full chip functionality, but at a much lower frequency. When you change the value of the divider, the resulting output frequency occurs immediately. The low-power dividers are controlled in the SCCR and their default division factors are one. For example, in a 50MHz system frequency, the SYNCCLK, LCDCLK, LCDCLK50, BRGCLK, and GCLKx are all 50MHz.

You can switch between the various low-power modes, as illustrated in Figure 5-18. Your software must set the appropriate CSRC and LPM fields in the PLPRCR and the POW bit in the MSR so the MPC823 can enter doze, sleep, or power-down mode from a normal mode. The MPC823 uses an interrupt to exit from any of these lower power modes. An enabled interrupt clears the LPM field, but does not change the CSRC bit. An interrupt switches automatically to normal high mode from normal low, doze high, doze low, sleep, or deep-sleep mode. Interrupts are generated by:

- Wake-up interrupts ($\overline{\text{IRQ}}$ signal) from the interrupt controller.
- Real-time clock, periodic interrupt timer, timebase, or decrementer interrupts.



NOTE: "&" DENOTES "AND" OPERATION, "|" DENOTES "OR" OPERATION, AND CPM_ACT DENOTES CPM ACTIVITY.
* SOFTWARE IS ACTIVE ONLY IN NORMAL HIGH/LOW MODES.
** TEXPS RECEIVES THE ZERO VALUE BY WRITING A ONE TO IT. WRITING A ZERO HAS NO EFFECT ON TEXPS.
*** YOU CAN SWITCH FROM NORMAL HIGH TO NORMAL LOW ONLY IF THE CONDITIONS TO AN INTERRUPT ARE CLEARED.

Figure 5-18. MPC823 Low-Power Mode Flowchart

The system responds quickly to an interrupt that does not come from the interrupt controller. The wake-up time from normal low, doze high, doze low, or sleep mode is between three and four VCOOUT clocks. For example, it could take 60-80ns to wake up in a 50MHz system. The level-sensitive interrupt from the interrupt controller is defined as a wake-up interrupt. It is only negated after the interrupt source bit is cleared. Any of the real-time clock, periodic interrupt timer, timebase, or decrementer interrupts can set the TMIST bit in the PLPRCR. The MPC823 clock module recognizes this interrupt as a pending interrupt when the TMIST bit is set. Therefore, the TMIST bit should be cleared before you enter any low-power mode other than normal high mode.

The wake-up time for all interrupt sources from the interrupt controller is measured in actual GCLK1 clocks. Once the interrupt is recognized, it takes between two and four GCLK1 clocks for the MPC823 to reach normal high mode. For example, it could take between 10.24µs and 20.48µs to wake up in a 50MHz system where DFNL = 111 (divided by 256). In normal and doze modes when the CSRC bit is set, the system toggles between the low and high frequencies. One of the following conditions must be met before you can switch from normal low mode to normal high mode.

- The communication processor module must be active (CPM_ACT)
- A pending interrupt from the interrupt controller must be recognized (INTERRUPT)
- The POW bit in the MSR must be cleared (normal operation) (POW)

If none of these conditions are met, the CSRC bit is set, and the interrupt status bits are reset, then the system automatically switches back to normal low mode. If the communication processor module is active, the system automatically switches from doze low mode to doze high mode. On the other hand, when the communication processor module is idle and the CSRC bit is set, then the system automatically switches back to doze low mode. A pending interrupt from the interrupt controller transfers the system from doze mode to normal high mode. The MPC823 exits deep-sleep mode and enters normal high mode when a wake-up interrupt from the interrupt controller, real-time clock, periodic interrupt timer, timebase, or decrementer is recognized.

In deep-sleep mode the SPLL is disabled and, therefore, the wake-up time from this mode is a maximum of 500 OSCM clocks. In 1-to-1 mode, the wake-up time can be a maximum of 1,000 EXTCLK clocks. For example, if the SPLL input frequency is 32kHz, the wake-up time is a maximum of 15.6ms and if it is 4MHz in 1-to-1 mode, the wake-up time is a maximum of 125µs.

To exit power-down mode and enter normal high mode, the $\overline{\text{HRESET}}$ pin must be asserted by external logic when the TEXP pin is asserted. The TEXPS bit in the PLPRCR is automatically set when a wake-up interrupt from the real-time clock, periodic interrupt timer, timebase, or decrementer occurs. $\overline{\text{HRESET}}$ should be asserted longer than the time it takes the power supply to wake up, plus the time it takes for the SPLL to reach lock condition. Another way to exit power-down mode is to assert $\overline{\text{HRESET}}$ when the TEXP pin is negated and the TEXPS bit is cleared. This causes the MPC823 to automatically assert the TEXP pin, which sets the TEXPS bit, and enter normal high mode.

When a timer expires, if enabled, the TEXP pin is asserted asynchronously with CLKOUT to show that the MPC823 is preparing to exit power-down mode. TEXP should be externally connected to a switch that should turn on the power supply to the chip, as illustrated in Figure 5-16.

In normal and doze modes, the system can be in the high mode defined by the DFNH field or in the low mode defined by the DFNL field. The MPC823 is in normal high mode after reset and this also the default state when the condition to exit low-power mode occurs. Table 5-7 provides the power consumption equations for each of these modes.

Table 5-7. MPC823 Low-Power Modes

OPERATION MODE	SPLL	CLOCKS	WAKE-UP METHOD	RETURN TIME FROM WAKE-UP EVENT TO NORMAL HIGH	MPC823 POWER CONSUMPTION AT 50MHZ	FUNCTIONALITY
Normal High LPM=00 TEXPS=1	Active	$VCOUT_{freq} \div 2^{DFNH}$	—	—	≅20 mW + $1/2^{DFNH}$ W	Full Functions Not In Use Are Shut Off
Normal Low LPM=00 TEXPS=1	Active	$VCOUT_{freq} \div 2^{DFNL+1}$	Software or Interrupt	Asynchronous Interrupts: 3-4 VCOOUT Clocks	≅20 mW + $1/2^{(DFNL+1)}$ W	
Doze High LPM=01 TEXPS=1	Active	$VCOUT_{freq} \div 2^{DFNH}$	Interrupt	Synchronous Interrupts: 2-4 GCLK1 Clocks	≅20mW + $0.4/2^{DFNH}$ W	Enabled: Real-Time Clock, Periodic Interrupt Timer, and Memory Controller: Disabled: Extended Core
Doze Low LPM=01 TEXPS=1	Active	$VCOUT_{freq} \div 2^{DFNL+1}$	Interrupt		≅20 mW + $0.4/2^{(DFNL+1)}$ W	
Sleep LPM=10 TEXPS=1	Active	Inactive	Interrupt	3-4 VCOOUT Clocks	<10 mW	Enabled: Real-Time Clock, Periodic Interrupt Timer, Timebase, and Decrementer
Deep-Sleep LPM=11 TEXPS=1	Inactive	Inactive	Interrupt	<500 OSCM Clocks 16ms-32kHz 125µs-4MHz	40µA	
Power-Down LPM=11 TEXPS=0	Inactive	Inactive	Interrupt	<500 OSCM Clocks + Power Supply Wake-Up (PwSp_Wake+ 16ms at 32kHz)	32kHz- ~10µA, KAPWR = 3.0V Temperature = 50° C	

Note: The communication processor module has its own power conservation logic, which it uses to automatically shut down its own clock when idle.

SECTION 6
THE POWERPC CORE

The MPC823 core is where the PowerPC™ architecture is implemented. It has the functionality of the PowerPC branch processor and fixed-point processor and includes all the PowerPC user mode (problem mode) instructions, except floating-point instructions, relevant privileged instructions, and all the registers associated with the processors and instructions. In addition, it contains part of the development support features of the MPC823, including breakpoint and watchpoint support, program flow tracking data generation, and debug mode operation in which the core is controlled by the development support system through the debug port module.

This section describes the functional specifications of the core. It is based on a document called the *PowerPC Microprocessor Family: The Programming Environment for 32-Bit Microprocessors* (MPCFPE32B/AD). Any reference to 64-bit implementation is not supported by this core. Only a subset of the PowerPC architecture books are supported, as indicated in **Appendix B MPC823 Instruction Set**.

6.1 FEATURES

The following is a list of the core's main features:

- 32-bit PowerPC Architecture
- Single-Issue Integer Machine
- Variable Pipeline Depth Architecture Tailored to Instruction Complexity
- Fully Static Design
- Out-of-Order Execution Termination
- Branch Prediction for Prefetch
- 32 × 32-Bit General-Purpose Register File
- Precise Exception Model
- Extensive Debug/Testing Support

6.2 BASIC STRUCTURE OF THE CORE

To accomplish its tasks, the core is divided into the following subunits:

- **Sequencer Unit**—Consists of the branch processor (next address generation), the instruction prefetch queue, and the interrupt handling mechanism. It controls some data structures within the register unit.
- **Register Unit**—Consists of all the user-visible registers, the register's scoreboard mechanism, and a history of previous operations to allow for a precise interrupt model. This module is physically split so that each data structure is implemented near the area in which it is used.
- **Fixed-Point Unit**—Implements all fixed-point instructions, except load/store instructions. This module is subdivided into the following two blocks:
  - IMUL/IDIV–Fixed-point multiply and divide instruction implementation.
  - ALU/BFU–Fixed-point logic, add, and subtract instruction implementation, as well as the bit field instructions.
- **Load/Store Unit**—Implements all load and store instructions. No floating-point processor load and store instructions are implemented.

6.2.1 Instruction Flow Within the Core

When fetched, instructions enter the instruction queue and enable branch folding by allowing out-of-order branch execution. Nonbranch instructions reaching the top of the instruction queue are issued to the execution units. Instructions can be flushed from the instruction queue in case of an exception on a previous instruction, interrupt, or miss-predicted fetch.

All instructions, including branches, enter the history buffer along with processor state information that can be affected by the instruction's execution. This information is used to enable out-of-order completion of instructions together with precise exception handling. When exceptions or interrupts occur, instructions can be flushed or recovered from the machine. The instruction queue is always flushed when the history buffer is recovered. An instruction retires from the machine after it finishes executing without exception and all preceding instructions are retired from the machine. Figure 6-1 illustrates the core's microarchitecture.

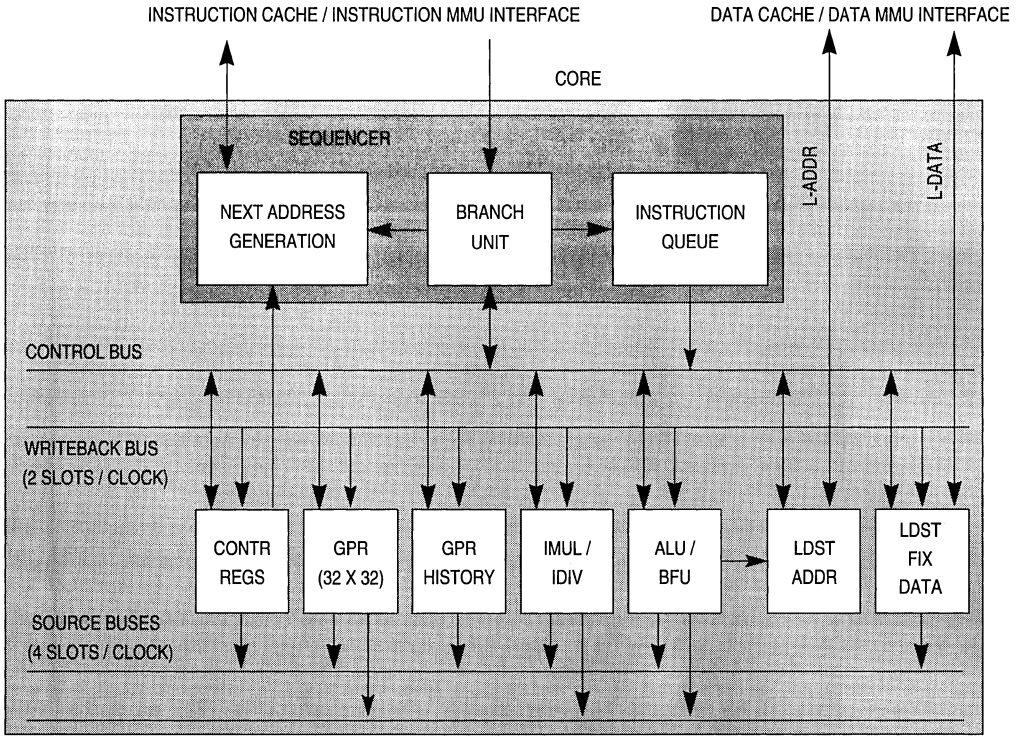


Figure 6-1. Block Diagram of the Core

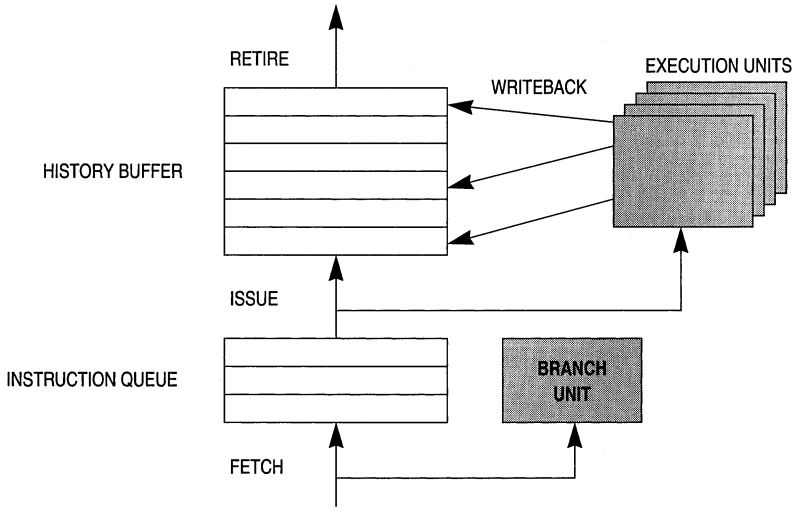


Figure 6-2. Instruction Flow Conceptual Diagram

6.2.2 Basic Instruction Pipeline

Figure 6-3 illustrates the basic instruction pipeline timing.

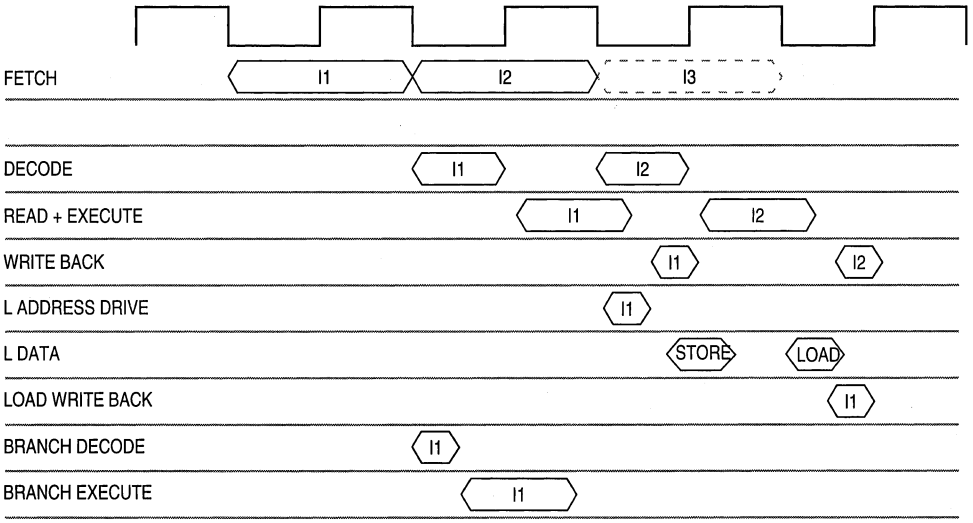


Figure 6-3. Basic Instruction Pipeline Timing Diagram

6.3 SEQUENCER UNIT

The instruction sequencer is the heart of the core. It controls data flow among execution units and register files, implements the basic instruction pipeline, fetches instructions from the memory system and issues them to available execution units, and maintains a state history so it can back up the machine in the event of an exception. The sequencer data path is illustrated in Figure 6-4. In addition, the sequencer implements all branch processor instructions, including flow control and condition register instructions.

6.3.1 Flow Control

Flow control operations, or branches disrupt normal instruction pipeline flow. A change in program flow creates bubbles in the pipeline because of the time it takes to fetch the newly targeted instruction stream. In typical code, with 4 or 5 sequential instructions between branches, the machine could waste up to 25% of its execution bandwidth waiting on branch latency.

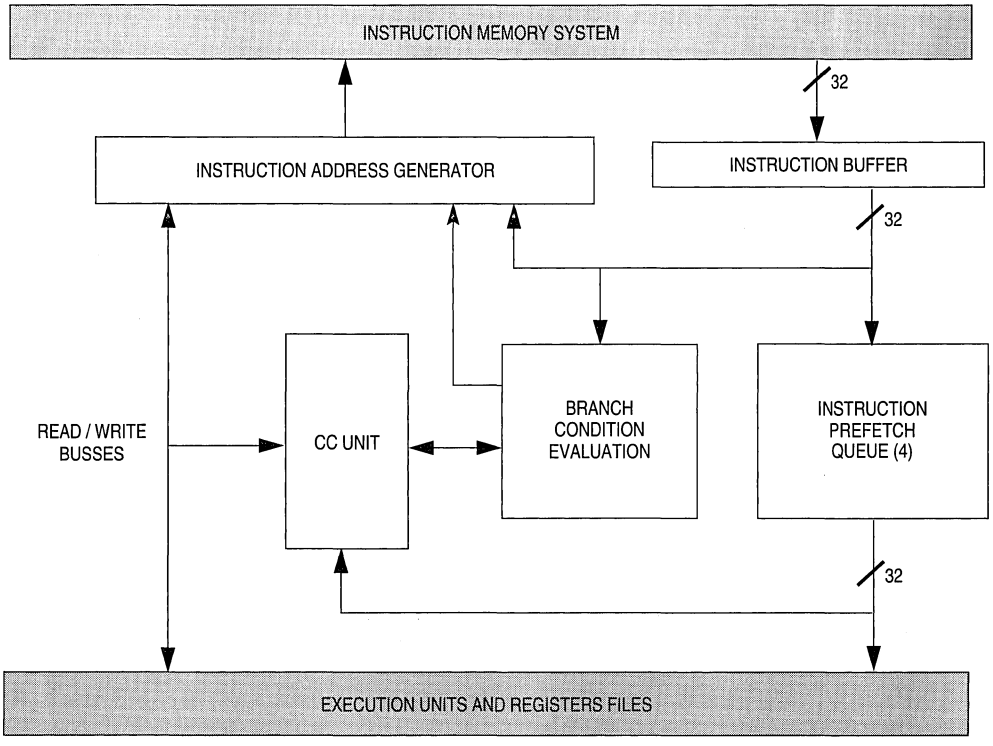


Figure 6-4. Sequencer Data Path

The sequencer maintains a 4-instruction deep instruction prefetch queue to execute branches in parallel with the execution of sequential instructions. Ideally, a sequential instruction is issued every clock, even when branches are present in the code. This is referred to as branch folding. The instruction prefetch queue also eliminates stalls due to long latency instruction fetches and all instructions are fetched into the instruction prefetch queue, but only sequential instructions are issued to the execution units when they reach the head of the queue. Branches enter the queue to mark watchpoints. See **Section 20 Development Capabilities and Interface** for details. Since branches do not prevent the issue of sequential instructions unless they come in pairs, the performance impact of entering branches in the instruction prefetch queue is negligible.

In addition to branch folding, the core implements a branch reservation station and static branch prediction so branches can issue as early as possible. The reservation station allows a branch instruction to issue even before its condition is ready. With the branch issued and out of the way, instruction prefetch can continue while the branch operand is being computed and the condition is evaluated. Static branch prediction determines which instruction stream is prefetched while the branch is being resolved. When the branch operand becomes available, it is forwarded to the branch unit and the condition is evaluated.

Branch instructions whose condition is unavailable and forced to issue to the reservation station are said to be predicted and these branches, which later turn out to have followed the wrong path, are said to be mispredicted. Branch instructions that issue with source data already available are unpredicted and those instructions fetched under a predicted branch are conditionally fetched. The core ignores conditionally prefetched instructions fetched under a mispredicted branch.

Table 6-1. Branch Prediction Policy

BRANCH TYPE	DEFAULT PREDICTION (Y=0)	MODIFIED PREDICTION (Y=1)
BC With Negative Offset	Taken	Fall Through
BC With Positive Offset	Fall Through	Taken
BCLR or BCCTR (lk or ctr) Address Ready	Fall Through	Taken
BCLR or BCCTR (lk or ctr) Address NOT Ready	Wait	Wait
B (Unconditional Branch)	Taken	Taken

6.3.2 Issuing Instructions

The sequencer tries to issue a sequential instruction on each clock whenever possible. However, for an instruction to issue, the execution unit must be available and able to discern whether or not the required source data is available and to ensure that no other instruction still in execution targets the same destination register. The sequencer informs the execution units of the existence of the instruction on the instruction bus. The execution units then decode the instruction, interrogate the register unit (if the operands and targets are free), and inform the sequencer that it accepts the instruction for execution.

6.3.3 Interrupts

The core interrupts can be generated when an exception occurs. An exception results when certain instructions are executed or an asynchronous external event occurs. There are five exception sources in the MPC823:

- External interrupt request
- Certain memory access conditions (protection faults and bus error)
- Internal errors, such as an attempt to execute an unimplemented opcode or floating-point arithmetic overflow
- Trap instructions
- Internal exceptions (breakpoints and debug counter's expiration)

Interrupt handling is transparent to user mode code and uses the same mechanism to handle all types of exceptions. When a user mode instruction experiences an exception, the machine is placed into privileged state and control is transferred to a software exception handler routine located at some offset within a memory-based vector table. Each interrupt generated in the machine transfers control to a different address in the vector table. For more information on initializing the base address of the vector table, refer to Table 6-6 as well as the PowerPC definition of the machine state register. When the exception has been handled, the handler can continue executing your program without ever knowing that an event has occurred. As specified in the *PowerPC Microprocessor Family: The Programming Environment for 32-Bit Microprocessors*, the core implements a precise interrupt model. This means that when an interrupt is taken, the following conditions are met:

- No instruction that logically follows the faulting instruction in the code stream has started executing.
- All instructions preceding the faulting instruction appear to have completed with respect to the executing processor.
- The precise location (address) of the faulting instruction is known to the exception handler.
- The instruction causing the exception might not have started executing (before interrupt), could be partially completed, or has completed (after interrupt), depending on the interrupt and instruction types. See Table 6-2 for details.

In any case, a partially completed instruction is restartable and can be reexecuted after the interrupt is handled. This precise exception model can simplify and speed up exception processing because the software does not have to manually save the machine's internal pipeline states, unwind the pipelines, or cleanly terminate the faulting instruction stream. Nor does it have to reverse the process to resume execution of the faulting stream.

Table 6-2. Before and After Interrupts

INTERRUPT TYPE	INSTRUCTION TYPE	BEFORE / AFTER	CONTENTS OF SRR0
Hard Reset	Any	NA	Undefined
System Reset	Any	Before	Next Instruction to Execute
Machine Check Interrupt	Any	Before	Faulting Instruction
Implementation Specific Instruction / Data TLB Miss / Error Interrupts	Any	Before	Faulting Fetch or Load/Store
Other Asynchronous Interrupts (Noninstruction Related Interrupts)	Any	Before	Next Instruction to Execute
Alignment Interrupt	Load / Store	Before	Faulting Instruction
Privileged Instruction	Any Privileged Instruction	Before	Faulting Instruction
Trap	tw, twi	Before	Faulting Instruction
System Call Interrupt	sc	After	Next Instruction to Execute
Trace	Any	After	Next Instruction to Execute
Debug I- Breakpoint	Any	Before	Faulting Instruction
Debug L- Breakpoint	Load / Store	After	Faulting Instruction + 4
Implementation Dependent Software Emulation Interrupt	NA	Before	Faulting Instruction
Floating-Point Unavailable	Floating-Point	Before	Faulting Instruction

6.3.4 Implementing the Precise Exception Model

To achieve maximum performance, many pieces of the instruction stream are concurrently processed by the core independent of the sequence specified by the executing program. Instructions execute in parallel and are completely random. The hardware ensures that this out-of-order operation never has an effect different than that specified by the program. This requirement is most difficult to assure when an interrupt occurs after instructions that logically follow the faulting instruction or have already completed. At the time of an interrupt, the machine state becomes visible to other processes and, therefore, must be in the appropriate architecturally specified condition. The core takes care of this in the hardware by automatically backing up the machine to the instruction which caused the interrupt and is, therefore, said to implement a precise exception model. This is, of course, assuming that the instruction causing the exception has not begun when the interrupt occurs.

To recover from an interrupt, a history buffer is used. This buffer is a FIFO queue that records the relevant machine state at the time of each instruction issue. Instructions are placed on the tail of the queue when they are issued and percolated to the head of the queue while they are in execution. Instructions remain in the queue until they complete execution and all preceding instructions have been completed to a point where no exception can be generated (in the core, such a condition is fulfilled by waiting for full completion).

In the event of an exception, the machine state necessary to recover the architectural state is available. As instructions finish executing, they are released (retired) from the queue and the buffer storage is reclaimed for new instructions entering the queue. An exception can be detected at any time during instruction execution and is recorded in the history buffer when the instruction finishes execution. The exception is not recognized until the faulting instruction reaches the head of the history queue, but once the exception is recognized, an interrupt process begins. The queue is reversed and the machine is restored to its state at the time the instruction is issued. Machine state is restored at a maximum rate of two floating-point and two fixed-point instructions per clock.

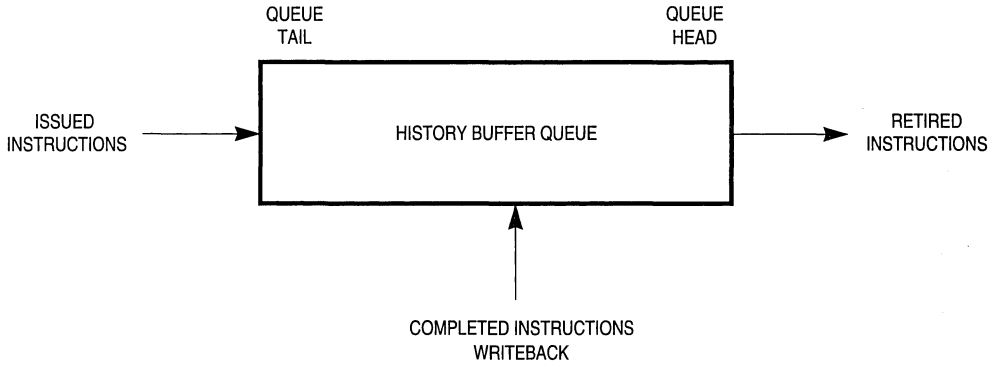


Figure 6-5. History Buffer Queue

To correctly restore the architectural state, the history buffer must record the value of the destination before the instruction is executed. The destination of a store instruction, however, is in memory and it is not practical from a performance standpoint to always read memory before writing it. Therefore, stores issue immediately to store buffers, but do not update memory until all previous instructions have finished executing without exception or the store has reached the head of the history buffer.

The history buffer has enough storage to hold the state of six instructions, but no more than four fixed-point instructions. The other two instructions can be condition code or branch instructions. In the event of a long latency instruction, it is possible that issued instructions will fill the history buffer. If so, instruction issue waits until the long latency operation finishes. The following types of instructions can potentially cause the history buffer to fill:

- Floating-point arithmetic instructions
- Integer divide instructions
- Instructions that affect or use resources external to the core (load/store instructions)

6.3.4.1 RESTARTABILITY AFTER AN INTERRUPT

Most of the interrupt cases in the core are always restartable. Some interrupts may be unable to be restarted because they can be recognized when the machine status save/restore 0 and 1 registers (SRR0 and SRR1) are busy. Such interrupts in the PowerPC architecture are known as system reset and machine check.

All other interrupt types defined in the architecture should always be restartable. By convention, no interrupt generating instruction should be executed between the start of an interrupt handler and the save of the registers altered by any interrupt or between restore of these registers and the execution of the **rfi** instruction. These registers being the SRR0 and SRR1 registers or the data address register (DAR) and data storage interrupt status register (DSISR) for some interrupt types.

External interrupts are also masked in these areas. In the core, two implementation-specific interrupt types can have this characteristic—debug port unmaskable interrupt and breakpoint interrupt in nonmaskable mode. Since there might be a situation in which it is preferable to be restartable, such as in the mentioned implementation-specific interrupts, a mechanism is defined to notify the interrupt handler code whether it is in a restartable state.

The mechanism uses a bit within the machine state register (MSR) called the recoverable interrupt bit (MSR_{RI}). The MSR_{RI} shadow bit in the SRR1 register indicates if the interrupt is restartable or not. This bit does not need to be checked on interrupt types that are restartable by convention, except those previously mentioned. The MSR_{RI} bit follows a similar behavior as the external interrupt enable bit (MSR_{EE}). Every time an interrupt occurs, MSR_{RI} is copied to its shadow in the SRR1 register (like the MSR) and cleared. Every time an **rfi** instruction is executed, MSR_{RI} is copied from its shadow in the SRR1 register. In addition, it can be altered by the software via the **mtmsr** instruction. The MSR_{RI} bit is intended to be set by the interrupt handler software after saving the machine state, (registers SRR0, SRR1, DAR, and DSISR if needed) and cleared by the interrupt handler software before retrieving the machine state.

In critical code sections where MSR_{EE} is negated but the SRR0 and SRR1 registers are not busy, MSR_{RI} should be left asserted. In these cases, if an interrupt occurs it is restartable. To facilitate the software manipulation of the MSR_{RI} and MSR_{EE} bits, the core includes special commands implemented as move to special register. The following table defines these special register mnemonics. A write of any data to these locations performs the operation specified in the following table. Any read from these locations is treated like any other unimplemented instruction and, therefore, results in an implementation-dependent software emulation interrupt.

Table 6-3. Special Ports to Machine State Register Bits

MNEMONIC	SPR	MSR_{EE}	MSR_{RI}	USED FOR
EIE	80	1	1	External Interrupt Enable: End of Interrupt Handler's Prologue, Enable Nested External Interrupts; End of Critical Code Segment in Which External Interrupts Were Disabled
EID	81	0	1	External Interrupt Disable, But Other Interrupts Are Recoverable: End of Interrupt Handler's Prologue, Keep External Nested Interrupts Disabled; Start of Critical Code Segment in Which External Interrupts Are Disabled
NRI	82	0	0	Nonrecoverable Interrupt: Start of Interrupt Handler's Epilogue

6.3.5 Processing an Interrupt

The following table provides the significant events that occur when processing an interrupt.

Table 6-4. Interrupt Latency

TIME POINT	FETCH	ISSUE	INSTRUCTION COMPLETE	KILL PIPELINE
A		Faulting Instruction Issue		
B			Instruction Complete and All Previous Instructions Complete	
C	Start Fetch Handler			Kill Pipeline
D ≤ B + 3 Clocks				
E		First Instruction of Handler Issued		

NOTES:
1. At time point A an instruction that will cause an interrupt is issued.
2. At time point B the excepting instruction has reached the head of the history queue, thus implying that all instructions preceding it in the code stream have finished execution without generating any interrupt. Also, the excepting instruction itself has completed execution. At this time the exception is "recognized" and exception processing begins. If, at this point, the instruction had not generated an exception, it would have been retired.
3. At time point C the sequencer starts to fetch the interrupt handler's first instruction.
4. By time point D the state of the machine prior to the issue of the excepting instruction is restored (the machine is restored to its state at time.
5. At time point E the machine state register and instruction pointer of the executing process have been saved and control has been transferred to the interrupt handler routine.

At time point A the excepting instruction issues and begins executing. During the interval between A and B, previously issued instructions are finishing execution. This interval is equivalent to the time required for all instructions currently in progress to complete. At time point B, the exception is recognized and during the interval between B and D the machine state is being restored. This time is a maximum of 10 cycles. At time point C, the core starts fetching the first instruction of the exception handler if the interrupt handler is external. It is 5 cycles if it is in the instruction cache and NO SHOW mode is on.

At time point D all state has been restored and during the interval between D and E, the machine is saving context information in the SRR0 and SRR1 registers, disabling interrupts, placing the machine in privileged mode, and continues the process of fetching the first instructions of the interrupt handler from the vector table. The interval between D and E requires a minimum of one clock. The interval between C and E depends on the memory system and is the time it takes to fetch the first instruction of the interrupt handler. For full history buffer restore time, it is no less then two clocks.

6.3.6 Serialization

The core has multiple execution units, each of which can be executing different instructions at the same time. This is normally transparent to your program, but in some special circumstances (debugging, I/O control, multiprocessor synchronization) it might become necessary to force the machine to serialize. There are two possible serialization actions defined for the core:

- Execution serialization—Instruction issue is halted until all instructions currently in progress have completed execution, all internal pipeline stages and instruction buffers have emptied, and all outstanding memory transactions are completed.
- Fetch serialization—Instruction fetch is halted until all instructions currently in the processor have completed execution. After fetch serialization, the machine is completely synchronized.

An attempt to issue a serializing instruction causes the machine to serialize before the instruction issues. Only the **sync** instruction guarantees serialization across PowerPC implementations. Fetching an **isync** instruction causes fetch serialization. Also, when the serialize mode bit ($CTRL_{SER}$) is asserted or is in debug mode, any instruction can cause fetch serialization.

6.3.6.1 LATENCY

The time required to serialize the machine is also the amount of time needed to complete the instructions currently in progress. This time is heavily dependent on the instructions in progress and the memory system latency. It is impossible to put an absolute upper bound on this time because the memory system design is not under the core's control. The time to complete the current instruction is generally the machine serialization time and the specific instruction execution time determines how long serialization takes. This can be either divide, load, or store a multiple, string, or pair of simple load/store instructions.

6.3.7 The External Interrupt

The core provides one external interrupt line: the architectural maskable external interrupt. In the MPC823, this interrupt is generated by the on-chip interrupt controller. It is software acknowledged and maskable by the MSR_{EE} bit, which is automatically cleared by the hardware to disable external interrupts when any interrupt is taken.

6.3.7.1 LATENCY

When an external interrupt is detected, every instruction that can retire from the history buffer does so and the interrupt is assigned to the instruction at the head of the history buffer (at point B in Table 6-4). However, the following conditions must be met before the instruction at the head of the queue can retire.

- The instruction must be completed without exception
- The instruction must either be a **mtspr**, **mtmsr**, or **rfi** instruction, a memory reference, or a storage or cache control instruction.

Any instruction that does not meet these criteria is discarded with all of its side effects and the execution at the end of the interrupt handler resumes with the first instruction that was discarded. If all the instructions in the history buffer were allowed to complete, execution at the end of the interrupt handler resumes with the next instruction. External interrupt latency depends on the time required to reference memory. The measurement is equal to the time taken for one of the following three possible events, in addition to the interval from B to E as shown in Table 6-4.

Longest load/store multiple/string instruction used

or

One bus cycle for aligned access

or

Two bus cycles for unaligned access

Actual system-level interrupt latency can be worse than just the interval from B to E. If the instruction prior to the one in which the interrupt was assigned generates an exception, then the exception is recognized first. If minimal interrupt latency is an important system parameter, interrupt handlers should save the machine context and reenable external interrupt as rapidly as possible so that a pending external interrupt receives service quickly.

6.3.8 Interrupt Ordering

There are two main types of interrupts:

- Instruction-related interrupts
- Asynchronous interrupts

Instruction-related exceptions (interrupt causes) are detected while the instruction is in various stages of being processed by the core. Exceptions found early in instruction processing preclude detection of further exceptions. This earlier interrupt will eventually be taken. If more than one instruction in the pipeline causes an exception, only the first exception is taken and causes an interrupt. The remaining instruction-induced exceptions are ignored. The following table lists the instruction-related interrupts in the order of detection within the instruction processing.

Table 6-5. Instruction-Related Interrupt Detection Order

NUMBER	INTERRUPT TYPE	CAUSE
1	Trace	Trace Bit Asserted[1]
2	Implementation Dependent Instruction TLB Miss	Instruction Memory Management Unit TLB Miss
3	Implementation Dependent Instruction TLB Error	Instruction Memory Management Unit Protection / Translation Error
4	Machine Check Interrupt	Fetch Error
5	Debug I- Breakpoint	Match Detection
6	Implementation Dependent Software Emulation Interrupt	Attempt to Invoke Unimplemented Feature
1	Floating-Point Unavailable	Attempt to is Made to Execute Floating-Point Instruction and MSR_{FP} =0
7[2]	Privileged Instruction	Attempt to Execute Privileged Instruction in Problem Mode
	Alignment Interrupt	Load/Store Checking
	System Call Interrupt	SC Instruction
	Trap	Trap Instruction
8	Implementation Dependent Data TLB Miss	Data Memory Management Unit TLB Miss
9	Implementation Dependent Data TLB Error	Data Memory Management Unit TLB Protection/ Translation Error
10	Machine Check Interrupt	Load or Store Access Error
11	Debug L- Breakpoint	Match Detection

NOTES:
1. The trace mechanism is implemented by letting one instruction go as if no trace is enabled and trapping the second instruction. This, of course, refers to this second instruction.
2. Exclusive for any one instruction.

More than one asynchronous interrupt cause or exception can be present at any time. However, when more than one interrupt causes exist, only the highest priority interrupt is taken, as shown in the following table.

Table 6-6. Interrupt Priority Mapping

NUMBER	INTERRUPT TYPE	CAUSE
1	Development Port Nonmaskable Interrupt	Signal From the Development Port
2	System Reset	NMI_L Assertion
3	Instruction-related Interrupts	Instruction Processing
4	Peripheral Breakpoint Request or Development Port Maskable Interrupt	Breakpoint Signal From Any Peripheral
5	External Interrupt	Signal From the Interrupt Controller
6	Decrementer Interrupt	Decrementer Request

6.4 THE REGISTER UNIT

The fixed-point registers bank holds thirty-two 32-bit fixed-point registers and some control registers. The register unit holds the general register files of the core and performs the following operations:

- Decodes the operand fields of all sequential instructions
- Drives the operand buses, as requested by the execution unit
- Performs scoreboard checking and signing
- Samples the resulting data from the writeback bus

6.4.1 Control Registers

The following tables describe the core control registers, also known as special-purpose registers, implemented within the MPC823.

Table 6-7. Standard Special-Purpose Registers

SPR			REGISTER NAME	PRIVILEGED	SERIALIZE ACCESS
DECIMAL	SPR 5:9	SPR 0:4			
1	00000	00001	XER	No	Write: Full Sync Read: Sync Relative to Load/ Store Operations
8	00000	01000	LR	No	No
9	00000	01001	CTR	No	No
18	00000	10010	DSISR	Yes	Write: Full Sync Read: Sync Relative to Load/ Store Operations
19	00000	10011	DAR	Yes	Write: Full Sync Read: Sync Relative to Load/ Store Operations
22	00000	10110	DEC	Yes	Write
26	00000	11010	SRR0	Yes	Write
27	00000	11011	SRR1	Yes	Write
272	01000	10000	SPRG0	Yes	Write
273	01000	10001	SPRG1	Yes	Write
274	01000	10010	SPRG2	Yes	Write
275	01000	10011	SPRG3	Yes	Write
287	01000	11111	PVR	Yes	No (Read-Only Register)

Table 6-8. Standard Timebase Register Mapping

SPR			REGISTER NAME	PRIVILEGED	SERIALIZE ACCESS
DECIMAL	SPR 5:9	SPR 0:4			
268	01000	01100	TBL Read[2]	No	Write - As a Store
269	01000	01101	TBU Read[2]	No	Write - As a Store
284	01000	11100	TBL Write[3]	Yes	Write - As a Store
285	01000	11101	TBU Write[3]	Yes	Write - As a Store

NOTES:
1. Extended opcode for **mftb**, 371 rather then 339.
2. Any write (**mtspr**) to this address results in an implementation-dependent software emulation interrupt.
3. Any read (**mftb**) to this address results in an implementation-dependent software emulation interrupt.

Table 6-9. Additional Special-Purpose Registers

SPR			REGISTER NAME	PRIVILEGED	SERIALIZE ACCESS
DECIMAL	SPR 5:9	SPR 0:4			
80	00010	10000	EIE[1]	Yes	Write
81	00010	10001	EID	Yes	Write
82	00010	10010	NRI	Yes	Write
144	00100	10000	CMPA[1]	Debug[3]	Fetch Sync on Write
145	00100	10001	CMPB	Debug	Fetch Sync on Write
146	00100	10010	CMPC	Debug	Fetch Sync on Write
147	00100	10011	CMPD	Debug	Fetch Sync on Write
148	00100	10100	ICR	Debug	Fetch Sync on Write
149	00100	10101	DER	Debug	Fetch Sync on Write
150	00100	10110	COUNTA	Debug	Fetch Sync on Write
151	00100	10111	COUNTB	Debug	Fetch Sync on Write
152	00100	11000	CMPE	Debug	Write: Fetch Sync Read: Synch Relative to Load/Store Operations
153	00100	11001	CMPF	Debug	Write: Fetch Sync Read: Synch Relative to Load/Store Operations
154	00100	11010	CMPG	Debug	Write: Fetch Sync Read: Synch Relative to Load/Store Operations
155	00100	11011	CMPH	Debug	Write: Fetch Sync Read: Synch Relative to Load/Store Operations
156	00100	11100	LCTRL1	Debug	Write: Fetch Sync Read: Synch Relative to Load/Store Operations
157	00100	11101	LCTRL2	Debug	Write: Fetch Sync Read: Synch Relative to Load/Store Operations
158	00100	11110	ICTRL	Debug	Fetch Sync on Write
159	00100	11111	BAR	Debug	Write: Fetch Sync Read: Synch Relative to Load/Store Operations
630	10011	10110	DPDR	Debug	Read and Write
631	10011	10111	DPIR[4]	Yes	Fetch

Table 6-9. Additional Special-Purpose Registers (Continued)

SPR			REGISTER NAME	PRIVILEGED	SERIALIZE ACCESS
DECIMAL	SPR 5:9	SPR 0:4			
638	10011	11110	IMMR	Yes	Write - As a Store
560	10001	10000	IC_CST	Yes	Write - As a Store
561	10001	10001	IC_ADR	Yes	Write - As a Store
562	10001	10010	IC_DAT	Yes	Write - As a Store
568	10001	11000	DC_CST	Yes	Write - As a Store
569	10001	11001	DC_ADR	Yes	Write - As a Store
570	10001	11010	DC_DAT	Yes	Write - As a Store
784	11000	10000	MI_CTR	Yes	Write - As a Store
786	11000	10010	MI_AP	Yes	Write - As a Store
787	11000	10011	MI_EPN	Yes	Write - As a Store
789	11000	10101	MI_TWC (MI_L1DL2P)	Yes	Write - As a Store
790	11000	10110	MI_RPN	Yes	Write - As a Store
816	11001	10000	MI_CAM	Yes	Write - As a Store
817	11001	10001	MI_RAM0	Yes	Write - As a Store
818	11001	10010	MI_RAM1	Yes	Write - As a Store
792	11000	11000	MD_CTR	Yes	Write - As a Store
793	11000	11001	M_CASID	Yes	Write - As a Store
794	11000	11010	MD_AP	Yes	Write - As a Store
795	11000	11011	MD_EPN	Yes	Write - As a Store
796	11000	11100	M_TWB (MD_L1P)	Yes	Write - As a Store
797	11000	11101	MD_TWC (MD_L1DL2P)	Yes	Write - As a Store
798	11000	11110	MD_RPN	Yes	Write - As a Store
799	11000	11111	M_TW (M_SAVE)	Yes	Write - As a Store
824	11001	11000	MD_CAM	Yes	Write - As a Store
825	11001	11001	MD_RAM0	Yes	Write - As a Store
826	11001	11010	MD_RAM1	Yes	Write - As a Store

NOTES:
1. See **Section 6.3.4.1 Restartability After An Interrupt**.
2. Refer to **Section 20.6.2 Development Port Registers**.
3. Protection of registers with "debug" privileges is described in **Section 20.6.1 Protecting the Development Port Registers.**
4. This register is a fetch-only register. Using **mtspr** is ignored and using **mfspr** gives an undefined value.

Table 6-10. Other Control Registers

DESCRIPTION	NAME	COMMENTS	PRIVILEGED	SERIALIZE ACCESS
Machine State Register	MSR	—	Yes	Write Fetch Sync
Condition Register	CR	—	No	Only **mtcrf**

6.4.1.1 PHYSICAL LOCATION OF SPECIAL REGISTERS

Some of the special registers in the core are physically located outside of the core. Access to these registers is gained the same way as any other special register—via the appropriate **mtspr**, **mfspr** instructions through the internal chip buses. Apart from the PowerPC timebase counter and decrementer, in the current implementation the following encoding is reserved for special registers not located within the core.

Table 6-11. Encoding Special Registers Located Outside the Core

SPR		RESERVED FOR
$SPR_{5:9}$	$SPR_{0:4}$	
100xx 110xx	xxxxx	Registers External to the Core
1x0xx	x0xxx	Reserved
10011	x0xxx	System Interface Unit Internal Registers
0xxxx	xxxxx	The Internal Bus Signifying Decrementer or Timebase
10000	x0xxx	Reserved
10000	x1xxx	Reserved
1100x	x0xxx	Instruction Memory Management Unit Implementation-Specific Control
1100x	x1xxx	Data Memory Management Unit Implementation-Specific Control
10001	x00xx	Instruction Cache Registers
10001	x10xx	Data Cache Registers

For these registers, a bus cycle is performed on the internal bus with the following address.

0:17	18:22	23:27	28:31
0. .0	$spr_{0:4}$	$spr_{5:9}$	0000

If any address error occurs on this cycle, an implementation-dependent software emulation interrupt is taken.

6.4.1.2 PowerPC STANDARD CONTROL REGISTER BIT ASSIGNMENT

6.4.1.2.1 Machine State Register. The 32-bit machine state register (MSR) defines the state of the processor. It can be read by the **mfmsr** instruction. However, it can be modified by the **mtmsr**, **sc**, and **rfi** instructions, as well as the hard reset configuration word. Refer to **Section 4.3.1.1 Hard Reset Configuration Word** for more information.

MSR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED													POW	RES	ILE
RESET	0													0	0	0
R/W	R/W													R/W	R/W	R/W
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	EE	PR	FP	ME	FE0	SE	BE	FE1	RES	IP	IR	DR	RESERVED		RI	LE
RESET	0	0	0	0	0	0	0	0	0	—	0	0	—		0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W

NOTE: — = Undefined.

Bits 0–12—Reserved

These bits are reserved and should be set to 0. Bits 0, 5, and 9 are loaded from the corresponding bit in the MSR when an interrupt is taken. The appropriate bit in the MSR is loaded from this bit when an **rfi** is executed. Reserved bits in the MSR are set from the source value on write and return the value last set for it on read.

POW—Power Management Disable

When this bit is clear it allows you to automatically switch between low and high frequency operation or between normal low mode and normal high mode. When this bit is set, power management is disabled. Refer to **Section 5 Clocks and Power Control** for more information on bus power management.

Bit 14—Reserved

This bit is reserved and should be set to 0.

ILE—Interrupt Little-Endian Mode

When an exception occurs, this bit is copied into the MSR to select the endian mode for the context established by the exception.

0 = Little-endian mode is selected.
1 = Big-endian mode is selected.

EE—External Interrupt Enable

This bit is loaded from the corresponding bit in the MSR when an interrupt is taken. The appropriate bit in the MSR is loaded from this bit when an **rfi** is executed.

PR—Problem State

This bit is loaded from the corresponding bit in the MSR when an interrupt is taken. The appropriate bit in the MSR is loaded from this bit when an **rfi** is executed.

FP—Floating-Point Available

This bit is loaded from the corresponding bit in the MSR when an interrupt is taken. The appropriate bit in the MSR is loaded from this bit when an **rfi** is executed.

ME—Machine Check Enable

This bit is loaded from the corresponding bit in the MSR when an interrupt is taken. The appropriate bit in the MSR is loaded from this bit when an **rfi** is executed.

FE0—Floating-Point Exception Mode 0

This bit is loaded from the corresponding bit in the MSR when an interrupt is taken. The appropriate bit in the MSR is loaded from this bit when an **rfi** is executed.

SE—Single-Step Trace Enable

This bit is loaded from the corresponding bit in the MSR when an interrupt is taken. The appropriate bit in the MSR is loaded from this bit when an **rfi** is executed.

BE—Branch Trace Enable

This bit is loaded from the corresponding bit in the MSR when an interrupt is taken. The appropriate bit in the MSR is loaded from this bit when an **rfi** is executed.

FE1—Floating-Point Exception Mode 1

This bit is loaded from the corresponding bit in the MSR when an interrupt is taken. The appropriate bit in the MSR is loaded from this bit when an **rfi** is executed.

Bit 24—Reserved

This bit is reserved and should be set to 0. It is loaded from the corresponding bit in the MSR when an interrupt is taken. The appropriate bit in the MSR is loaded from this bit when an **rfi** is executed. Reserved bits in the MSR are set from the source value on write and return the value last set for it on read.

IP—Interrupt Prefix

This bit is loaded from the corresponding bit in the MSR when an interrupt is taken. The reset value of this bit is determined by the hard reset configuration word. For more information, see **Section 4.3.1.1 Hard Reset Configuration Word**.

0 = The interrupt prefix is 0x000n_nnnn.
1 = The interrupt prefix is 0xFFFn_nnnn.

IR—Instruction Relocate

This bit is loaded from the corresponding bit in the MSR when an interrupt is taken. The appropriate bit in the MSR is loaded from this bit when an **rfi** is executed.

DR—Data Relocate

This bit is loaded from the corresponding bit in the MSR when an interrupt is taken. The appropriate bit in the MSR is loaded from this bit when an **rfi** is executed.

Bits 28 and 29—Reserved

These bits are reserved and should be set to 0. Reserved bits in the MSR are set from the source value on write and return the value last set for it on read.

RI—Recoverable Interrupt

This bit is loaded from the corresponding bit in the MSR when an interrupt is taken. The appropriate bit in the MSR is loaded from this bit when an **rfi** is executed.

LE—Little-Endian Mode

This bit is loaded from the corresponding bit in the MSR when an interrupt is taken. The appropriate bit in the MSR is loaded from this bit when an **rfi** is executed. This bit is loaded from the ILE bit when an interrupt is taken

6.4.1.2.2 The Condition Register. The condition register (CR) contains eight 4-bit condition fields. Each field can have one of the following formats and the software can assign an arbitrary meaning to them.

- Bit 0—Negative (LT). The result is negative.
- Bit 1—Positive (GT). The result is positive.
- Bit 2—Zero (EQ). The result is zero.
- Bit 3—Summary Overflow (SO). The values of this bit is copied from XER_{SO}.

6.4.1.2.3 Fixed-Point Exception Cause Register. The following table provides the bit assignments for the fixed-point exception cause register (XER). The bits are based on the final result produced by executing an instruction.

XER

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	SO	OV	CA	RESERVED												
RESET	0	0	0	0												
R/W	R/W	R/W	R/W	R/W												
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED									BCNT						
RESET	0									0						
R/W	R/W									R/W						

SO—Summary Overflow

This bit is set when an instruction other than **mtspr** sets the OV bit. Once it is set, it stays that way until an **mtspr** or **mcrxr** instruction clears it. It is not altered by compare instructions or other instructions that cannot overflow. SO is cleared and OV is set when an **mtspr** instruction is executed, which supplies the zero values for these bits.

OV—Overflow

When this bit is set it indicates that an overflow has occurred during the execution of an instruction. The add, subtract, and negate instructions with OE equal to 1, set this bit if the carry out of the MSB is not equal to that carry out of the MSB+1. The multiply low and divide instructions with OE equal to 1, set this bit if the result cannot be represented in 64 bits (**mulld**, **divd**, **divdu**) or 32 bits (**mullw**, **divw**, **divwu**). The OV bit is not altered by compare instructions that cannot overflow, except **mtspr** to the XER and **mcrxr**.

CA—Carry

This bit is set during execution of the following instructions. It is not altered by compare instructions or by other instructions that cannot carry, except shift right algebraic, **mtspr** to the XER, and **mcrxr**.

- Add carrying, subtract from carrying, add extended, and subtract from extended instructions set CA if there is a carry out of the MSB. Otherwise, it is cleared.
- The shift right algebraic instructions set CA if any 1 bits have been shifted out of a negative operand. Otherwise, it is cleared.

Bits 3–24—Reserved

These bits are reserved and should be set to 0.

BCNT—Byte Count for Load/Store String Operations

This field specifies the number of bytes to be transferred by a **lswx** or **stswx** instruction.

6.4.1.3 INITIALIZING THE CONTROL REGISTERS

6.4.1.3.1 System Reset Interrupt. A system reset interrupt occurs when the $\overline{\text{IRQ0}}$ pin is asserted. The only control registers affected by the system reset interrupt are the MSR, SRR0, and SRR1 registers. For information on the values of these registers, refer to **Section 7.3.7.3.1 System Reset Interrupt**.

6.4.1.3.2 Hard/Soft Reset. When a hard or soft reset occurs, the registers affected by system reset are set in the same way. The following list shows how each register is set.

- SRR0, SRR1—Set to an undefined value.
- MSR_{IP}—Programmable.
- MSR_{ME}—Set to zero.
- ICTRL—Set to 0.
- LCTRL1—Set to 0.
- LCTRL2—Set to 0.
- $COUNTA_{16-31}$—Set to 0.
- $COUNTB_{16-31}$—Set to 0.
- ICR—Set to 0 (no interrupt occurred).
- $DER_{2,14,28:31}$—Set to 1 (all debug-specific interrupts cause debug mode entry).

6.5 THE FIXED-POINT UNIT

The fixed-point unit implements all fixed-point processor instructions, except the fixed-point storage access instructions that are implemented by the load/store unit. Refer to the *PowerPC Microprocessor Family: The Programming Environment for 32-Bit Microprocessors* manual for more information.

6.5.1 XER Update In Divide Instructions

The divide instructions have a relatively long latency, but those instructions can update the OV bit in the XER after one cycle. Therefore, data dependency on the XER is limited to one cycle, although the divide instruction latency can be a maximum of 11 clocks.

6.6 THE LOAD/STORE UNIT

The load/store unit handles all data transfers between the register file and chip internal bus. It is implemented as an independent execution unit so that stalls in the memory pipeline do not cause the master instruction pipeline to stall, unless there is a data dependency. The unit is fully pipelined so that memory instructions of any size can be issued on back-to-back cycles.

There is a 32-bit wide data path between the load/store unit and fixed-point register file. Single-word accesses to the internal on-chip data RAM require one clock, resulting in two clock latencies and double-word accesses require two clocks, which results in three clock latencies. Because the internal bus is 32 bits wide, double-word transfers take two bus accesses. The load/store unit implements all of the PowerPC load/store instructions in hardware, including unaligned and string accesses.

The following is a list of the load/store unit's main features:

- Supports many instructions
- Two-entry load/store instruction address queue
- Pipelined operation
- Minimal load latency–2 clocks using 1 clock on-chip data RAM
- Minimal store latency–1 clock since the load/store unit ends the store execution in 2 clocks, using 1 clock on-chip data RAM.
- Load/store multiple and string instructions synchronize
- Load/store breakpoint/watchpoint detection support

Figure 6-6 illustrates a conceptual block diagram of the load/store unit and its two queues. The address queue is a 2-entry queue shared by all load/store instructions and the fixed-point data queue is a 2-entry, 32-bit wide queue that holds fixed-point data.

The load/store unit has a dedicated writeback bus so that loaded data received from the internal bus is written directly back to the fixed- or floating-point register files.

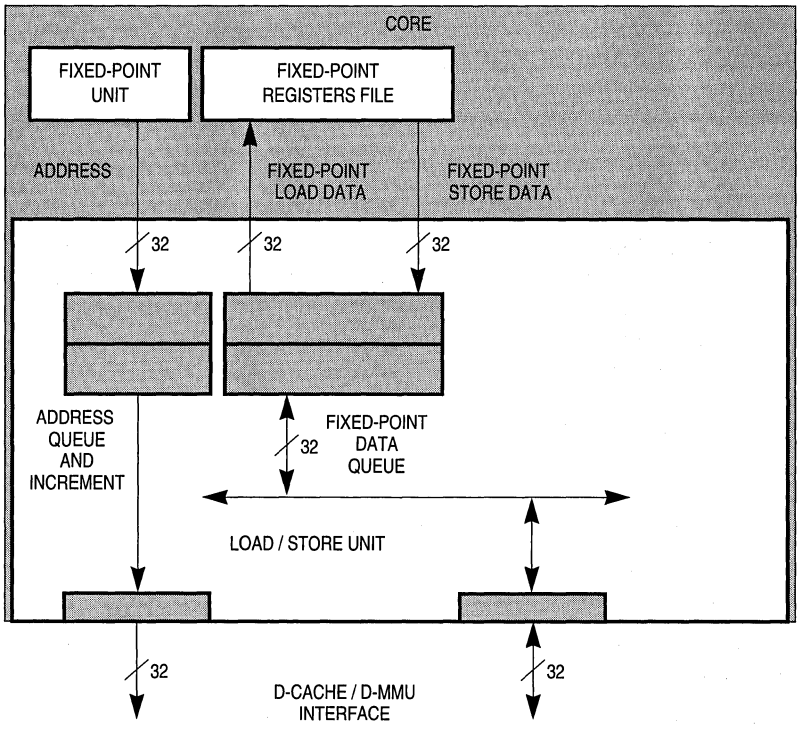


Figure 6-6. Load/Store Unit Functional Block Diagram

To execute multiple (**lmw, stmw**) instructions, string instructions, unaligned accesses, and double-precision floating-point load/store instructions, the load/store unit contains an address incrementor that generates the needed addresses. This allows the unit to execute the unaligned accesses without stalling the master instruction pipeline.

6.6.1 Issuing Load/Store Instructions

When load or store instructions are encountered, the load/store unit checks the scoreboard to determine if all of the operands are available. These operands include:

- Address register operands
- Source data register operands (for store instructions)
- Destination data register operands (for load instructions)
- Destination address register operands (for load/store with update instructions)

If all operands are available, the load/store unit takes the instruction and enables the sequencer to issue a new instruction. Then, using a dedicated interface, the load/store unit notifies the integer unit of the need to calculate the effective address. All load/store instructions are executed and terminated in order.

If there are no prior instructions waiting in the address queue, the load/store instruction is issued to the data cache immediately at the time the instruction is taken. Otherwise, if there are prior instructions remaining whose addresses have not yet been issued to the data cache, the instruction is inserted into the address queue and data is inserted into the respective store data queue. For load/store with update instructions, the destination address register is written back on the following clock, regardless of the address queue's state.

6.6.2 Serializing Load/Store Instructions

The following load/store instructions are not taken until all previous instructions have terminated.

- Load/store multiple instructions—**lmw**, **stmw**
- Storage synchronization instructions—**lwarx**, **stwcx**, **sync**
- String instructions—**lswi**, **lswx**, **stswi**, **stswx**
- Move to internal special registers and move to off-core special registers

The following load/store instructions must terminate before more instructions can be issued.

- Load/store multiple instructions—**lmw**, **stmw**
- Storage synchronization instructions—**lwarx**, **stwcx**, **sync**
- String instructions—**lswi**, **lswx**, **stswi**, **stswx**

6.6.3 Instructions Issued to the Data Cache

The load/store unit pipelines load accesses. The individual cache cycles of all multiregister instructions and unaligned accesses are pipelined into the data cache interface.

6.6.4 Issuing Store Instruction Cycles

A new store instruction is not issued to the data cache until all prior instructions have terminated without an exception because the core supports the precise interrupt model. If a load instruction is followed by a store instruction, a one clock delay is inserted between the load bus cycle termination and the store cycle issue.

6.6.5 Issuing Nonspeculative Load Instructions

Load instructions targeted at a nonspeculative memory region are identified as nonspeculative one clock cycle after the previous load/store bus cycle termination, but only if all prior instructions have terminated normally and without an exception. The nonspeculative identification relates to the state of the cycle's associated instruction. For **lmw**, although the cycles are pipelined into the bus, they are all marked as nonspeculative because the instruction is nonspeculative.

With a single register load instruction where more than one bus cycle is generated, some of the cycles can be marked as speculative and later cycles can be marked as nonspeculative after all prior instructions terminate. When executing speculative load cycles to the nonspeculative external memory region, no external cycles are generated until the load instruction becomes nonspeculative.

6.6.6 Executing Unaligned Instructions

The load/store unit supports fixed-point unaligned accesses in the hardware. The 32-bit L-bus only supports naturally aligned transfers. For an unaligned instruction, the load/store unit breaks the instruction into a series of aligned transfers that are pipelined into the bus. Figure 6-7 illustrates the number of bus cycles needed to execute unaligned instructions.

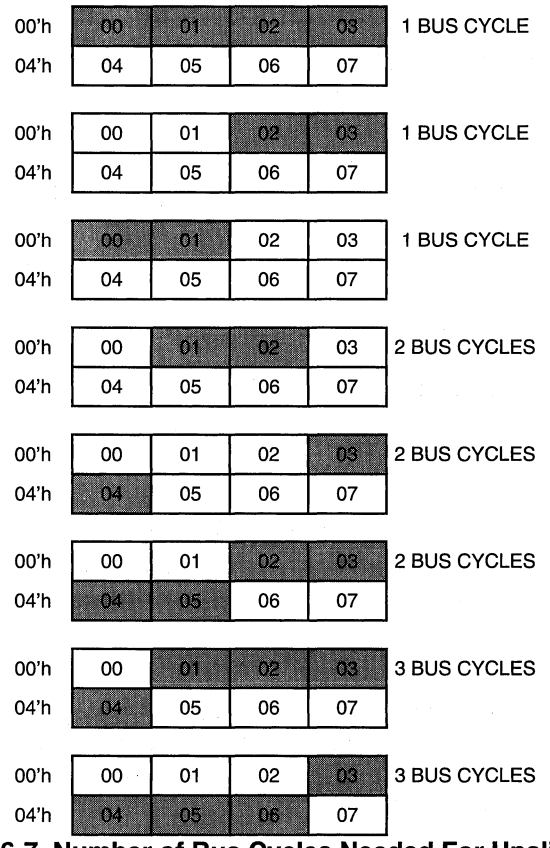


Figure 6-7. Number of Bus Cycles Needed For Unaligned, Single Register Fixed-Point Load/Store Instructions

6.6.7 Little-Endian Mode Support

The load/store unit implements the little-endian mode as it is specified by the PowerPC architecture and in this mode the modified address is issued to the data cache. For an individual scalar unaligned transfer or an attempted execution of a multiple/string instruction, an alignment exception is generated.

6.6.8 Atomic Update Primitives

The **lwarx** and **stwcx** instructions are atomic update primitives. Storage reservation accesses made by the same processor are implemented by the load/store unit. The external bus interface module implements storage reservation as it relates to accesses made by external bus masters. Accesses made by other internal masters to internal memories implements storage reservation as it relates to special internal bus snoop logic. This logic is implemented in the data cache.

When a **lwarx** instruction is executed the load/store unit issues a cycle to the data cache with a special attribute. For an external memory access, this attribute causes the external bus interface module to set a storage reservation on the cycle address. The external bus interface module is then responsible for snooping the external bus or getting an indication from external snoop logic if the storage reservation is broken by some other processor accessing the same location. When an **stwcx** instruction to external memory is executed, the external bus interface module checks to see if a reservation was lost. If loss of reservation has occurred, the cycle is blocked from going to the external bus and the external bus interface module notifies the load/store unit of a **stwcx** failure.

The MPC823 storage reservation supplies hooks for the support of storage reservation implementation in a hierarchical bus structure. For a full description of the storage reservation mechanism, refer to **Section 7 PowerPC Architecture Compliance**. In case of storage reservation on internal memory, a **lwarx** indication causes the on-chip snoop logic to latch the address. This logic notifies the load/store unit in the case of an internal master store access, then the reservation is reset. If a new **lwarx** instruction address phase is successfully executed it replaces any previous storage reservation address at the appropriate snoop logic. However, when an **stwcx** instruction is executed, the storage reservation is canceled, unless an alignment interrupt condition is detected.

6.6.9 Instruction Timing

The following table summarizes the different load/store instructions timing in the case of zero wait state memory references on a parked bus. With external memory accesses, pipelined external accesses are assumed.

Table 6-12. Load/Store Instructions Timing

INSTRUCTION TYPE	LATENCY		CLEARED FROM LOAD/STORE UNIT	
	DATA CACHE	EXTERNAL MEMORY	DATA CACHE	EXTERNAL MEMORY
Fixed-Point Single Target Register Load (Aligned)	2 Clocks	5 Clocks	2 Clocks	5 Clocks
Fixed-Point Single Target Register Store (Aligned)	1 Clock	1 Clock	2 Clocks	5 Clocks
Load/Store Multiple	1 + N	$3+N+\left(\frac{N+1}{3}\right)$	1 + N	$3+N+\left(\frac{N+1}{3}\right)$

NOTE: N denotes the number of registers transferred.

String instructions are broken into a series of aligned bus accesses. Figure 6-8 illustrates the maximum number of bus cycles needed for string instruction execution.

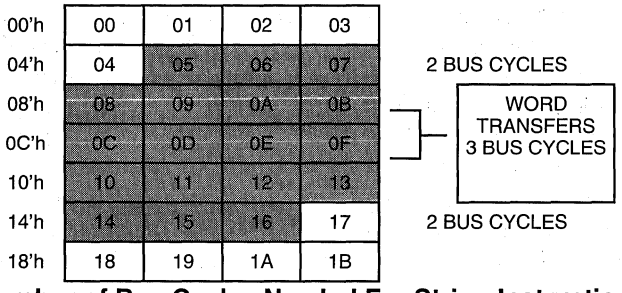


Figure 6-8. Number of Bus Cycles Needed For String Instruction Execution

6.6.10 Stalling Storage Control Instructions

A storage control instruction waits one clock before it is taken.

6.6.11 Accessing Off-Core Special Registers

Access to special registers—**mtspr** and **mfspr**—implemented off-core is executed by the load/store unit via the internal bus using a special cycle. Refer to **Section 6.4.1.1 Physical Location of Special Registers** for detailed information. If the access terminates in a bus error, then an implementation-dependent software emulation interrupt is taken. All write operations to off-core special registers (**mtspr**) are previously synchronized. In other words, the instruction is not taken until all prior instructions terminate.

6.6.12 Storage Control Instructions

Cache management instructions and lookaside buffer management instructions are implemented by the load/store unit. These instructions are implemented as the special bus write cycles, which are issued to the data cache interface.

6.6.13 Exceptions

6.6.13.1 DAR, DSISR, AND BAR OPERATION. The load/store unit keeps track of all instructions and bus cycles. When a bus error occurs, the data address register (DAR) is loaded with the cycle's effective address. For a multicycle instruction, the effective address of the first offending cycle is loaded.

The data storage interrupt status register (DSISR) notifies the error when an exception caused by the load/store occurs. For a memory management unit error, this register is loaded with the error status delivered by the memory management unit. For other exceptions, the DSISR is loaded with the instruction information as defined by the PowerPC architecture for alignment exception. The breakpoint address register (BAR) notifies the address on which an L-bus breakpoint occurred. For a multicycle instruction, the BAR contains the address of the first cycle with which the breakpoint condition was associated. The BAR has a valid value only when a data breakpoint interrupt is taken. At any other time, its value is boundedly undefined. The following situations cause the DAR, BAR, and DSISR registers to be updated.

Table 6-13. Value Summary of the DAR, BAR, and DSISR Registers

INTERRUPT TYPE	DAR VALUE	DSISR VALUE	BAR VALUE
Data Storage Interrupt	Cycle EA	Memory Management Unit Error Status	Undefined
Alignment Interrupt	Data EA	Instruction Information	Undefined
L-Bus Breakpoint Interrupt	Does Not Change	Does Not Change	Cycle EA
Machine Check Interrupt	Cycle EA	Instruction Information	Undefined
Implementation Dependent Software Emulation Interrupt	Does Not Change	Does Not Change	Undefined
Floating-Point Unavailable Interrupt	Does Not Change	Does Not Change	Undefined
Program Interrupt	Does Not Change	Does Not Change	Does Not Change

SECTION 7
POWERPC ARCHITECTURE COMPLIANCE

This section describes implementation-dependent choices made for the core on issues that are optional on the PowerPC™ architecture as defined in the *PowerPC Architecture Books I, II, and III*. It also describes features that exist in the architecture, but are not supported by the core. The information in this section is based on the PowerPC books, but you can also refer to the *PowerPC Microprocessor Family: The Programming Environment for 32-Bit Microprocessors* (MPCFPE32B/AD) manual for more information.

7.1 POWERPC USER INSTRUCTION SET ARCHITECTURE (BOOK I)

7.1.1 Computation Modes

The core is a 32-bit fixed-point implementation of the PowerPC architecture. Any reference in the *PowerPC Architecture Books I, II, and III* regarding 64-bit implementations are not supported by this core. No floating point of the architecture is implemented.

7.1.2 Reserved Fields

Reserved fields in instructions are described under the specific instruction definition sections. Unless otherwise stated in the specific instruction description, fields marked I, II, and III in the instruction are discarded by the core decoding. Thus, this type of invalid form instructions yield results of the defined instructions with the appropriate field zero. In most cases, the reserved fields in registers are ignored on write and return zeros for them on read for any control register implemented by the core. Exceptions to this rule are bits 16-23 of the fixed-point exception cause register (XER) and the reserved bits of the machine state register (MSR), which are set by the source value on write and return the value last set for it on read.

7.1.3 Classes of Instructions

Nonoptional instructions (except floating-point load, store, and compute instructions) are implemented by the hardware. Optional instructions are executed by implementation-dependent code and any attempt to execute one of these commands causes the core to take the implementation-dependent software emulation interrupt (offset x'01000' of the vector table). Illegal and reserved instruction class instructions are supported by implementation-dependent code and, thus the core hardware generates the implementation-dependent software emulation interrupt. How the core treats invalid and preferred instruction forms is described in the specific processor compliance sections.

7.1.4 Exceptions

Invocation of the system software for any exception caused by an instruction in the core is precise, regardless of the type and setting.

7.1.5 The Branch Processor

7.1.6 Fetching Instructions

The core fetches a number of instructions into its internal buffer (the instruction prefetch queue) prior to execution. If a program modifies the instructions it intends to execute, it should call a system library program to ensure that the modifications are visible to the instruction fetching mechanism prior to executing the modified instructions.

7.1.7 Branch Instructions

The core implements all the instructions defined for the branch processor by the *PowerPC User Instruction Set Architecture (Book I)* in the hardware. For details about the performance of various instructions, see Table 8-1 of this manual.

7.1.7.1 INVALID BRANCH INSTRUCTION FORMS. Bits marked with *z* in the BO encoding definition are discarded by the core decoding. Thus, these types of invalid form instructions yield results of the defined instructions with the *z* bit zero. If the decrement and test CTR option is specified for the **bcctr** or **bcctrl** instructions, the target address of the branch is the new value of the CTR. Condition is evaluated correctly, including the value of the counter after decrement.

7.1.7.2 BRANCH PREDICTION. The core uses the *y* bit to predict path for prefetch. Prediction is only done for not-ready branch conditions. No prediction is done for branches to the link or count register if the target address is not ready (see Table 6-1 for more details).

7.1.8 The Fixed-Point Processor

The core implements the following fixed-point instructions:

- Arithmetic instructions
- Compare instructions
- Trap instructions
- Logical instructions
- Rotate and shift instructions
- Move to/from system register instructions

All hardware instructions are defined for the fixed-point processor in the *PowerPC User Instruction Set Architecture (Book I)*. For details about the performance of the various instructions, see Table 8-1 of this manual.

7.1.8.1 MOVE TO/FROM SYSTEM REGISTER INSTRUCTIONS. Move to/from invalid special registers in which $\mathbf{spr_0}$ =1 invokes the privilege instruction error interrupt handler if the processor is in problem state (user mode). For a list of all implemented special registers, refer to **Section 6.4.1 Control Registers**.

7.1.8.2 FIXED-POINT ARITHMETIC INSTRUCTIONS. If you try to perform any of the following divisions using the **divw[o][.]** instruction

0x80000000 ÷ -1

<anything> ÷ 0

then, the contents of RT are 0x80000000 and if Rc =1, the contents of the bits in the CR field 0 are LT = 1, GT = 0, EQ = 0, and SO is set to the correct value.

In the **cmpi**, **cmp**, **cmpli**, and **cmpl** instructions, the L bit is applicable for 64-bit implementations. For the MPC823, if L = 1 the instruction form is invalid. The core ignores this bit and, therefore, the behavior when L = 1 is identical to the valid form instruction with L = 0.

7.1.9 The Load/Store Processor

The load/store processor supports all of the 32-bit implementation fixed-point PowerPC load/store instructions in the hardware.

7.1.9.1 FIXED-POINT LOAD WITH UPDATE AND STORE WITH UPDATE INSTRUCTIONS. For load with update and store with update instructions where RA =0, the EA is written into R0. For load with update instructions where RA = RT, RA is boundedly undefined.

7.1.9.2 FIXED-POINT LOAD AND STORE MULTIPLE INSTRUCTIONS. For these types of instructions, EA must be a multiple of four. If it is not, the system alignment error handler is invoked. For a **lmw** instruction (if RA is in the range of registers to be loaded), the instruction completes normally. RA is then loaded from the memory location as follows:

RA <- MEM(EA+(RA-RT)*4, 4)

7.1.9.3 FIXED-POINT LOAD STRING INSTRUCTIONS. Load string instructions behave the same as load multiple instructions, with respect to invalid format in which RA is in the range of registers to be loaded. If RA is in the range, it is updated from memory.

7.1.9.4 STORAGE SYNCHRONIZATION INSTRUCTIONS. For these type of instructions, EA must be a multiple of four. If it is not, the system alignment error handler is invoked.

7.1.9.5 OPTIONAL INSTRUCTIONS. No optional instructions are supported.

7.1.9.6 LITTLE-ENDIAN BYTE ORDERING. The load/store unit supports little-endian byte ordering as specified in the *PowerPC User Instruction Set Architecture (Book I).* In little-endian mode, if an attempt is made to execute an individual scalar unaligned transfer, as well as a multiple or string instruction, an alignment interrupt is taken.

7.2 PowerPC VIRTUAL ENVIRONMENT ARCHITECTURE (BOOK II)

7.2.1 Storage Model

The MPC823 caches are structured as follows:

- Physically addressed split 2K instruction cache and 1K data cache
- Two-way set associative managed with LRU replacement algorithm
- 16-byte (4 words) line size with one valid bit per line

7.2.1.1 MEMORY COHERENCE. Hardware memory coherence is not supported in the MPC823 hardware, but can be performed in the software or by defining storage as cache inhibited. In addition, the MPC823 does not provide any data storage attributes to an external system.

7.2.1.2 ATOMIC UPDATE PRIMITIVES. Both the **lwarx** and **stwcx** instructions are implemented according to the PowerPC architecture requirements. When the storage accessed by the **lwarx** and **stwcx** instructions is in the cache-allowed mode, it is assumed that the system works with the single master in this storage region. Therefore, if a data cache miss occurs, the access on the internal and external buses does not have a reservation attribute.

The MPC823 does not cause the system data storage error handler to be invoked if the storage accessed by the **lwarx** and **stwcx** instructions is in the writethrough required mode. Also, the MPC823 does not provide support for snooping an external bus activity outside the chip. The provision is made to cancel the reservation inside the MPC823 by using the CR_B and KR_B input pins. Data cache has a snoop logic to monitor the internal bus for communication processor module accesses of the address associated with the last **lwarx** instruction.

7.2.2 The Effect Of Operand Placement on Performance

The load/store unit hardware supports all of the PowerPC load/store instructions. An optimal performance can be obtained for naturally aligned operands. These accesses result in optimal performance for a maximum size of four bytes and good performance for double precision floating-point operands. Unaligned operands are supported in the hardware and are broken into a series of aligned transfers. The effect of operand placement on performance is as stated in the *PowerPC Virtual Environment Architecture (Book II),* except for the case of 8-byte operands. Because the MPC823 uses a 32-bit wide data bus, the performance is good rather than optimal. Refer to **Section 6.6.6 Executing Unaligned Instructions** for a description of fixed-point unaligned instruction execution and timing and to **Section 6.6.9 Instruction Timing** for a description of string instruction timing.

7.2.3 The Storage Control Instructions

The MPC823 interprets the cache control instructions (**icbi**, **isync**, **dcbt**, **dcbi**, **dcbf**, **dcbz**, **dcbst**, **eieio**, and **dcbtst**) as if they pertain only to the MPC823 cache. These instructions do not broadcast. Any bus activity caused by these instructions is what happens when an operation is performed on the MPC823 cache.

- Instruction Cache Block Invalidate (**icbi**)—The effective address is translated by the memory management unit and the associative block in the instruction cache is invalidated if hit.
- Instruction Synchronize (**isync**)—The **isync** instruction waits for all previous instructions to complete and then discards any prefetched instructions, thus causing subsequent instructions to be fetched or refetched from memory and executed.
- Data Cache Block Touch (**dcbt**)—The block associated with this instruction is checked for hit in the cache. If it is a miss, the instruction is treated as a regular miss, except that the bus error does not cause an interrupt. If no error occurs, the line is written into the cache.
- Data Cache Block Touch for Store (**dcbtst**)—The block associated with this instruction is checked for a hit in the cache. If it is a miss, the instruction is treated as a regular miss, except that bus error does not cause an interrupt. If no error occurs, the signal is written into the cache.
- Data Cache Block Set to Zero (**dcbz**)—This instruction is executed according to how it is defined in the *PowerPC Virtual Environment Architecture Book II.*
- Data Cache Block Store (**dcbst**)—This instruction is executed according to how it is defined in the *PowerPC Virtual Environment Architecture Book II.*
- Data Cache Block Invalidate (**dcbi**)—The effective address is translated by the memory management unit and the associative block in the data cache is invalidated if hit.
- Data Cache Block Flush (**dcbf**)—This instruction is executed according to how it is defined in the *PowerPC Virtual Environment Architecture Book II.*
- Enforce In-Order Execution of I/O (**eieio**)—When executing an **eieio** instruction, the load/store unit waits until all previous accesses have terminated before issuing cycles associated with load/store instructions after the **eieio** instruction.

7.2.4 Timebase

A description of the timebase register can be found in **Section 12 System Interface Unit** and **Section 5 Clocks and Power Control**.

7.3 POWERPC OPERATING ENVIRONMENT ARCHITECTURE (BOOK III)

The MPC823 has an internal memory space that includes memory-mapped control registers and memory that is used by various modules on the chip. This memory is part of the main memory as seen by the core but cannot be accessed by any external system master.

7.3.1 The Branch Processor

7.3.1.1 MACHINE STATE REGISTER. The floating-point exception mode is ignored by the MPC823. The IP bit initial state after reset is set as programmed by the reset configuration specified in **Section 12 System Interface Unit**.

7.3.1.2 PROCESSOR VERSION REGISTER. The value of the PVR register's version field is x'0050'. The value of the revision field is x'0000' and it is incremented each time that the software distinguishes between the core revisions.

7.3.1.3 BRANCH PROCESSORS INSTRUCTIONS. The core implements all the instructions defined for the branch processor in the *PowerPC User Instruction Set Architecture Book I* in the hardware. For the details about the performance of various instructions, see Table 8-1 of this manual.

7.3.2 The Fixed-Point Processor

7.3.2.1 UNSUPPORTED REGISTERS. The following registers are not supported by the MPC823. Refer to **Section 7.3.3 Storage Model** for more details.

SDR 1	IBAT2U	DBAT1U	IBAT0L	IBAT3L	DBAT2L
EAR	IBAT2L	DBAT1L	IBAT1U	DBAT0U	DBAT3U
IBAT0U	IBAT3U	DBAT2U	IBAT1L	DBAT0L	DBAT3L

7.3.2.2 ADDED REGISTERS. For a list of the added special purpose registers, see Table 6-9.

7.3.3 Storage Model

Page sizes are 4K, 16K, 512K, and 8M and an optional sub-page granularity of 1K for 4K pages in a maximum real memory size of 4G. Neither ordinary or direct-store segments are supported.

7.3.3.1 ADDRESS TRANSLATION. If address translation is disabled (MSR_{IR}=0 for instruction accesses or MSR_{DR}=0 for data accesses), the effective address is treated as the real address and is passed directly to the memory subsystem. Otherwise, the effective address is translated by using the translation lookaside buffer (TLB) mechanism of the memory management unit (MMU). Instructions are not fetched from no-execute or guarded storage and data accesses are not executed speculatively to or from the guarded storage.

The features of the MMU hardware is as follows:

- 32-entry fully associative instruction TLB
- 32-entry fully associative data TLB
- Supports up to 16 virtual address spaces
- Supports 16 access protection groups
- Supports fast software tablewalk mechanism

7.3.4 Reference and Change Bits

No reference bit is supported by the MPC823. However, the change bit is supported by using the data TLB error interrupt mechanism when writing to an unmodified page.

7.3.5 Storage Protection

Two main protection modes are supported by the MPC823:

- Domain manager mode
- PowerPC mode

For more details, refer to **Section 11 Memory Management Unit.**

7.3.6 Storage Control Instructions

7.3.6.1 DATA CACHE BLOCK INVALIDATE (dcbi). This instruction is executed according to the definition in *PowerPC Operating Environment Architecture (Book III).*

7.3.6.2 TLB INVALIDATE ENTRY (tlbie). This instruction is performed as defined by the architecture, except that the 22 most-significant bits of the EA are used for address compare.

7.3.6.3 TLB INVALIDATE ALL (tlbia). This instruction is performed as defined by the architecture.

7.3.6.4 TLB SYNCHRONIZE (tlbsync). This instruction is implemented like a regular **mtspr** instruction as it relates to engine synchronization with no further effects.

7.3.7 Interrupts

7.3.7.1 CLASSES. All interrupts associated with storage are implemented as precise interrupts by the core, which means that a load/store instruction is not complete until all possible error indications are sampled from the load/store bus. This also implies that a store or nonspeculative load instruction is not issued to the load/store bus until all previous instructions have completed. If a late error occurs, a store cycle (or a nonspeculative load cycle) can be issued and aborted.

7.3.7.2 PROCESSING. In each interrupt handler, when registers SRR0 and SSR1 are saved, MSR_{RI} can be set to 1.

7.3.7.3 DEFINITIONS. The following table defines the offset value by interrupt type and the sections that follow describe each interrupt in detail.

Table 7-1. Offset of First Instruction by Interrupt Type

OFFSET (HEX)	INTERRUPT TYPE
00000	Reserved
00100	System Reset
00200	Machine Check
00300	Data Storage
00400	Instruction Storage
00500	External
00600	Alignment
00700	Program
00800	Floating Point Unavailable
00900	Decrementer
00A00	Reserved
00B00	Reserved
00C00	System Call
00D00	Trace
00E00	Floating Point Assist
01000	Implementation-Dependent Software Emulation
01100	Implementation-Dependent Instruction TLB Miss
01200	Implementation-Dependent Data TLB Miss
01300	Implementation-Dependent Instruction TLB Error
01400	Implementation-Dependent Data TLB Error
01500 - 01BFF	Reserved
01C00	Implementation-Dependent Data Breakpoint
01D00	Implementation-Dependent Instruction Breakpoint
01E00	Implementation-Dependent Peripheral Breakpoint
01F00	Implementation-Dependent Nonmaskable Development Port

7.3.7.3.1 System Reset Interrupt. A system reset interrupt occurs when the $\overline{\text{IRQ0}}$ pin is asserted and the following registers are set. Execution begins at physical address 0x100 if the hard reset configuration word IIP bit is 1. Execution begins at physical address 0xFFF00100 if the hard reset configuration word IIP bit is 0.

SRR0—Save/Restore Register 0

Set to the effective address of the next instruction the processor executes if no interrupt conditions are present.

SRR1—Save/Restore Register 1

Used to save the machine status prior to exceptions and to restore status when an **rfi** instruction is executed.

1–4	Set to 0.
10–15	Set to 0.
Other	Loaded from bits 16-31 of the MSR. In the current implementation, Bit 30 of the SRR1 is never cleared, except by loading a zero value from MSR_{RI}.

MSR—Machine State Register

IP	No change.
ME	No change.
LE	Bit is copied from the ILE.
Other	Set to 0.

7.3.7.3.2 Machine Check Interrupt. A machine check interrupt indication is received from the U-bus as a response to the address or data phase. It is usually caused by one of the following conditions:

- The accessed address does not exist
- A data error is detected

As defined in *PowerPC Operating Environment Architecture (Book III)*, machine check interrupts are enabled when MSR_{ME}=1. If MSR_{ME} = 0 and a machine check interrupt indication is received, the processor enters the checkstop state. The behavior of the core in checkstop state is dependent on the working mode as defined in **Section 20.4.2.1 Debug Mode Enable vs. Debug Mode Disable**. When the processor is in debug mode enable, it enters the debug mode instead of the checkstop state. When in debug mode disable, instruction processing is suspended and cannot be restarted without resetting the core.

An indication that can generate an automatic reset in this condition is sent to the system interface unit. Refer to the **Section 12 System Interface Unit** for more details. If the machine check interrupt is enabled (MSR_{ME}=1) it is taken. If SRR1 Bit 30 =1, the interrupt is recoverable and the following registers are set.

SRR0—Save/Restore Register 0

Set to the effective address of the instruction that caused the interrupt.

SRR1—Save/Restore Register 1

1	Set to 1 for instruction fetch-related errors and 0 for load/store-related errors.
2–4	Set to 0.
10–15	Set to 0.
Other	Loaded from bits 16-31 of the MSR. In the current implementation, Bit 30 of the SRR1 is never cleared, except by loading a zero value from MSR_{RI}.

MSR—Machine State Register

IP	No change.
ME	Set to 0.
LE	Bit is copied from the ILE.
Other	Set to 0.

When using the load/store bus, the following registers are set:

DSISR—Data/Storage Interrupt Status Register

0–14	Set to 0.
15–16	Set to bits 29-30 of the instruction if X-form instruction and to 0b00 if D-form.
17	Set to Bit 25 of the instruction if X-form instruction and to Bit 5 if D-form.
18–21	Set to bits 21-24 of the instruction if X-form instruction and to bits 1-4 if D-form.
22–31	Set to bits 6-15 of the instruction.

DAR—Data Address Register

Set to the effective address of the data access that caused the interrupt.

Execution resumes at offset x'00200' from the base address indicated by MSR_{IP}.

7.3.7.3.3 Data Storage Interrupt. A data storage interrupt is never generated by the hardware. However, the software may branch to this location as a result of either an implementation-specific data TLB error or miss interrupt.

7.3.7.3.4 Instruction Storage Interrupt. An instruction storage interrupt is never generated by the hardware, but the software may branch to this location as a result of an implementation-specific instruction TLB error interrupt.

7.3.7.3.5 Alignment Interrupt. An alignment interrupt occurs as a result of one of the following conditions:

- The operand of a floating-point load or store is not word aligned.
- The operand of a load/store multiple is not word aligned.
- The operand of a **lwarx** or **stwcx** is not word aligned.
- The operand of a load/store individual scalar instruction is not naturally aligned when $MSR_{LE} = 1$.
- An attempt to execute a multiple/string instruction is made when $MSR_{LE} = 1$.

7.3.7.3.6 Program Interrupt. The MPC823 cannot generate a floating-point exception type interrupt. Likewise, an illegal instruction type program interrupt is not generated by the core, but an implementation-dependent software emulation interrupt is generated instead. A privileged instruction program interrupt is generated for an on-core valid special-purpose register (SPR) field or any SPR encoded as an external special register if SPR_0 =1 and MSR_{PR} =1, as well as if you try to execute privileged instruction occurred when MSR_{PR} =1. See Table 6-11 for details.

7.3.7.3.7 Floating-Point Unavailable Interrupt. The MPC823 cannot generate a floating-point exception type interrupt. An implementation-dependent software emulation interrupt will be taken when you try to execute floating-point instruction, regardless of MSR_{FP}.

7.3.7.3.8 Trace Interrupt. A trace interrupt occurs if MSR_{SE} = 1 and any instruction except **rfi** is successfully completed or if MSR_{BE} = 1 and a branch is completed. Notice that the trace interrupt does not occur after an instruction that causes an interrupt. The monitor/debugger software must change the vectors of other possible interrupt addresses to single-step these instructions. If this is unacceptable, other debug features can be used. Refer to **Section 20 Development Capabilities and Interface** for more information. The following registers are set on a trace interrupt:

SRR0—Save/Restore Register 0

Set to the effective address of the instruction following the executed instruction.

SRR1—Save/Restore Register 1

1–4	Set to 0.
10–15	Set to 0.
Other	Loaded from bits 16-31 of the MSR. In the current implementation, Bit 30 of the SRR1 is never cleared, except by loading a zero value from MSR_{RI}.

MSR—Machine State Register

IP	No change.
ME	No change.
LE	Bits are copied from the ILE.
Other	Set to 0.

Execution resumes at offset x'00D00' from the base address indicated by MSR_{IP}.

7.3.7.3.9 Floating-Point Assist Interrupt. The floating-point assist interrupt is not generated by the MPC823. An implementation-dependent software emulation interrupt will be taken when you try to execute a floating-point instruction.

7.3.7.3.10 Implementation-Dependent Software Emulation Interrupt. An implementation-dependent software emulation interrupt occurs as a result of one of the following conditions:

- When executing any unimplemented instruction, including all illegal and unimplemented optional and floating-point instructions.
- When executing a **mtspr** or **mfspr** that specifies an on-core unimplemented register, regardless of SPR_0.
- When executing a **mtspr** or **mfspr** that specifies an off-core unimplemented register and SPR_0=0 or MSR_{PR}=0 (no program interrupt condition). For more information, refer to **Section 7.3.7.3.6 Program Interrupt**.

In addition, the following registers are set:

SRR0—Save/Restore Register 0

Set to the effective address of the instruction that caused the interrupt.

SRR1—Save/Restore Register 1

1–4	Set to 0.
10–15	Set to 0.
Other	Loaded from bits 16-31 of the MSR. In the current implementation, Bit 30 of the SRR1 is never cleared, except by loading a zero value from MSR_{RI}.

MSR—Machine State Register

IP	No change.
ME	No change.
LE	Bits are copied from the ILE.
Other	Set to 0.

Execution resumes at offset x'01000' from the base address indicated by MSR_{IP}.

7.3.7.3.11 Implementation-Specific Instruction TLB Miss Interrupt. This type of interrupt occurs if MSR_{IR}=1 and you try to fetch an instruction from a page whose effective page number cannot be translated by TLB. The following registers are set:

SRR0–Save/Restore Register 0

Set to the effective address of the instruction that caused the interrupt.

SRR1—Save/Restore Register 1

0–3	Set to 0.
4	Set to 1.
10	Set to 1.
11–15	Set to 0.
Other	Loaded from bits 16-31 of the MSR. In the current implementation, Bit 30 of the SRR1 is never cleared, except by loading a zero value from MSR_{RI}.

MSR—Machine State Register

IP	No change.
ME	No change.
LE	Bits are copied from the ILE.
Other	Set to 0.

Some instruction TLB registers are set to the values described in **Section 11 Memory Management Unit**. Execution resumes at offset x'01100' from the base address indicated by MSR_{IP}.

7.3.7.3.12 Implementation-Specific Instruction TLB Error Interrupt. This type of interrupt occurs as a result of one of the following conditions:

- The effective address cannot be translated. Either the segment or page valid bit of this page is cleared in the translation table.
- The fetch access violates storage protection.
- The fetch access is to guarded storage and MSR_{IR}=1.

The following registers are set:

SRR0—Save/Restore Register 0

Set to the effective address of the instruction that caused the interrupt.

SRR1—Save/Restore Register 1

1	Set to 1 if the translation of an attempted access is not found in the translation tables. Otherwise, set to 0.
2	Set to 0.
3	Set to 1 if the fetch access was to a guarded storage when MSR_{IR} = 1 or when bit 4 is set. Otherwise, set to 0.
4	Set to 1 if the storage access is not permitted by the protection mechanism; otherwise set to 0. In the first revision when this bit is set, Bits 3 and 10 are also set, but in future revisions this bit may be set alone.
10	Set to 1 when Bit 4 is set. Otherwise, set to 0.
11–15	Set to 0.
Other	Loaded from bits 16-31 of the MSR. In the current implementation, Bit 30 of the SRR1 is never cleared, except by loading a zero value from MSR_{RI}.

MSR—Machine State Register

IP	No change.
ME	No change.
LE	Bits are copied from the ILE.
Other	Set to 0.

Some instruction TLB registers are set to a value described in **Section 11 Memory Management Unit**. Execution resumes at offset x'01300' from the base address indicated by MSR_{IP}.

7.3.7.3.13 Implementation-Specific Data TLB Miss Interrupt. This type of interrupt occurs when MSR_{DR}=1 and you try to access a page whose effective page number cannot be translated by TLB. The following registers are set:

SRR0—Save/Restore Register 0

Set to the effective address of the instruction that caused the interrupt.

SRR1—Save/Restore Register 1

1–4	Set to 0.
10–15	Set to 0.
Other	Loaded from bits 16-31 of the MSR. In the current implementation, Bit 30 of the SRR1 is never cleared, except by loading a zero value from MSR_{RI}.

MSR—Machine State Register

IP	No change.
ME	No change.
LE	Bits are copied from the ILE.
Other	Set to 0.

Some instruction TLB registers are set to the values described in **Section 11 Memory Management Unit**. Execution resumes at offset x'01200' from the base address indicated by MSR_{IP}.

7.3.7.3.14 Implementation-Specific Data TLB Error Interrupt. This type of interrupt occurs as a result of one of the following conditions:

- No effective address of a **load**, **store**, **icbi**, **dcbz**, **dcbst**, **dcbf** or **dcbi** instruction can be translated (either the segment or page valid bit of this page is cleared in the translation table).
- The access violates storage protection.
- An attempt was made to write to a page with a negated change bit.

The following registers are set:

SRR0—Save/Restore Register 0

Set to the effective address of the instruction that caused the interrupt.

SRR1—Save/Restore Register 1

1–4	Set to 0.
10–15	Set to 0.
Other	Loaded from bits 16-31 of the MSR. In the current implementation, Bit 30 of the SRR1 is never cleared, except by loading a zero value from MSR_{RI}.

MSR—Machine State Register

IP	No change.
ME	No change.
LE	Bits are copied from the ILE.
Other	Set to 0.

DSISR—Data/Storage Interrupt Status Register

0	Set to 0.
1	Set to 1 if the translation of an attempted access is not found in the translation tables. Otherwise, set to 0.
2–3	Set to 0.
4	Set to 1 if the storage access is not permitted by the protection mechanism; otherwise set to 0.
5	Set to 0.
6	Set to 1 for a store operation and to 0 for a load operation.
7–31	Set to 0.

DAR—Data Address Register

Set to the effective address of the data access that caused the interrupt.

Some instruction TLB registers are set to the values described in **Section 11 Memory Management Unit**. Execution resumes at offset x'01400' from the base address indicated by MSR_{IP}.

7.3.7.3.15 Implementation-Specific Debug Register. An implementation-specific debug interrupt occurs as a result of one of the following conditions:

- When there is an internal breakpoint match (for more details, refer to **Section 20.3 Generating Watchpoints And Breakpoints**).
- When a peripheral breakpoint request is presented to the interrupt mechanism.
- When the development port request is presented to the interrupt mechanism. Refer to **Section 20 Development Capabilities and Interface** for details on how to generate the development port request.

The following registers are set:

SRR0—Save/Restore Register 0

For I-breakpoints, set to the effective address of the instruction that caused the interrupt. For L-breakpoint, set to the effective address of the instruction following the instruction that caused the interrupt. For development port maskable request or a peripheral breakpoint, set to the effective address of the instruction that the processor would have executed next if no interrupt conditions were present. If the development port request is asserted at reset, the value of SRR0 is undefined.

SRR1—Save/Restore Register 1

1–4	Set to 0.
10–15	Set to 0.
Other	Loaded from bits 16-31 of the MSR. In the current implementation, Bit 30 of the SRR1 is never cleared, except by loading a zero value from MSR_{RI}.

If the development port request is asserted at reset, the value of SRR1 is undefined.

MSR—Machine State Register

IP	No change.
ME	No change.
LE	Bits are copied from the ILE.
Other	Set to 0.

For L-bus breakpoint instances, the following registers are set to:

BAR—Breakpoint Address Register

Set to the effective address of the data access as computed by the instruction that caused the interrupt.

DSISR—Data/Storage Interrupt Status Register

Do not change.

DAR—Data Address Register

Do not change.

The execution resumes from an address equal to the base indicated by the MSR_{IP} and the following offset.

- x'01D00'–For an instruction breakpoint match
- x'01C00'–For a data breakpoint match
- x'01E00'–For a development port maskable request or a peripheral breakpoint
- x'01F00'–For a development port nonmaskable request

7.3.7.4 PARTIALLY EXECUTED INSTRUCTIONS. In general, the architecture allows instructions to be partially executed when an alignment or data storage interrupt occurs. In the core, instructions are not executed if an alignment interrupt condition is detected or if a data storage interrupt is never generated by the hardware. In the MPC823, the instruction can be partially executed only in the case of load/store instructions that cause multiple access to the memory subsystem—multiple/string and unaligned load/store instructions.

In this instance, the instruction can be partially completed if one of the accesses (except the first one) causes a miss in the data TLB. The implementation-specific data TLB miss interrupt is taken in this case. For the update forms, the update register (RA) is not altered.

7.3.8 Timer Facilities

Descriptions of the timebase and decrementer registers can be found in **Section 12 System Interface Unit** and **Section 5 Clocks and Power Control**.

7.3.9 Optional Facilities and Instructions

Any other *PowerPC Operating Environment Architecture (Book III)* optional facilities and instructions that are not discussed here are not implemented by the MPC823 hardware. Any attempt to execute any of these instructions causes an implementation-dependent software emulation interrupt to occur.

SECTION 8
INSTRUCTION EXECUTION TIMING

This section describes the timing of the instruction cycles in terms of clock cycles, including serialization, latency, and blockage.

8.1 INSTRUCTION TIMING LIST

The following table lists the instruction execution timing in terms of latency and blockage of the appropriate execution unit. A serializing instruction has the effect of blocking all execution units.

Table 8-1. Instruction Execution Timing

INSTRUCTIONS	LATENCY	BLOCKAGE	EXECUTION UNIT	SERIALIZING INSTRUCTION
Branch Instructions: b, ba, bl, bla, bc, bca, bcl, bcla, bclr, bclrl, bcctr, bcctl	Taken 2	2	Branch Unit	No
	Not Taken 1	1		
System Call: sc, rfi	Serialize + 2	Serialize + 2	—	Yes
CR Logical: crand, crxor, cror, crnand, crnor, crandc, creqv, crorc, mcrf	1	1	CR Unit	No
Fixed-Point Trap Instructions: twi, tw	Taken Serialize + 3	Serialize + 3	ALU / BFU	After
	Not Taken 1	1		No
Move to Special Registers: mtspr, mtcrf, mtmsr, mcrxr Except mtspr to LR and CTR and External to the Core Registers	Serialize + 1	Serialize + 1	All	Yes
Move to LR, CTR: mtspr	1	1	Branch Unit	No
Move to External to the Core Special Registers: mtspr, mttb, mttbu	Serialize + 1[8]	Serialize + 1	LDST	Yes
Move from External to the Core Special Registers: mfspr, mftb, mftbu	Load Latency	1	LDST	No
Move from Special Registers Located Internal to the Core: mfspr[1]	1	1	—	See List [2]

Table 8-1. Instruction Execution Timing (Continued)

INSTRUCTIONS	LATENCY	BLOCKAGE	EXECUTION UNIT	SERIALIZING INSTRUCTION
Move from Others: mfcr, mfmsr	Serialize + 1	Serialize + 1	—	See List [3]
Fixed-Point Arithmetic: addi, add[o][.], addis, subf[o][.], addic, subfic, addic., addc[o][.], adde[o][.], subfc[o][.], subfe[o][.], addme[o][.], addze[o][.], subfme[o][.], subfze[o][.], neg[o][.]	1	1	ALU / BFU	No
Fixed-Point Arithmetic (Divide Instructions): divw[o][.], divwu[o][.]	Min 2 Max 11 [4]	Min 2 Max 11 [5]	IMUL / IDIV	No
Fixed-Point Arithmetic (Multiply Instructions): mul, mullw[o][.], mulhw[.], mulhwu[.]	2	1-2 [6]	IMUL / IDIV	No
Fixed Point Compare: cmpi, cmp, cmpli, cmpl	1	1	ALU / BFU	No
Fixed-Point Logical: andi., andis., ori, oris, xori, xoris, and[.], or[.], xor[.], nand[.], nor[.], eqv[.], andc[.], orc[.], extsb[.], extsh[.], cntlzw[.]	1	1	ALU / BFU	No
Fixed-Point Rotate and Shift: rlwinm[.], rlwnm[.], rlwimi[.], slw[.], srw[.], srawi[.], sraw[.]	1	1	ALU / BFU	No
Fixed-Point Load Instructions: lbz, lbzu, lbzx, lbzux, lhz, lhzu, lhzx, lhzux, lha, lhau, lhax, lhaux, lwz, lwzu, lwzx, lwzux, lhbrx, lwbrx.	2 [7]	1	LDST	No
Fixed-Point Store Instructions: stb, stbu, stbx, stbux, sth, sthu, sthx, sthux, stw, stwu, stwbrx, stwx, stwux, sthbrx	1 [8]	1	LDST	No
Fixed-Point Load and Store Multiple Instructions: lmw, smw	Serialize + 1 + Number of Registers	Serialize + 1 + Number of Registers	LDST	Yes
Synchronize: sync	Serialize + 1	Serialize + 1	LDST	Yes
Storage Synchronization Instructions: lwarx, stwcx.	Serialize + 2	Serialize + 2	LDST	Yes
Move Condition Register from XER: mcrxr	Serialize + 1	Serialize + 1	LDST	Yes (Before)
Move to / from Special Purpose Register (Debug, DAR, DSISR): mtspr, mfspr	Serialize + 1	Serialize + 1	LDST	Yes (Before)
String Instructions: lswi, lswx, stswi, stswx	Serialize + 1 + Number of Words Accessed	Serialize + 1 + Number of Words Accessed	LDST	Yes

Table 8-1. Instruction Execution Timing (Continued)

INSTRUCTIONS	LATENCY	BLOCKAGE	EXECUTION UNIT	SERIALIZING INSTRUCTION
Storage Control Instructions: isync	Serialize	Serialize	Branch	Yes
Order Storage Access: eieio	1	1	LDST	Next Load or Store is Synchronized Relative to All Prior Load or Store
Cache Control: icbi	1	1	LDST, I-Cache	No

NOTES:

1. Refer to Table 6-11 for details.
2. Refer to **Section 6.4.1 Control Registers.**
3. See Table 6-10 for details.
4. $$\text{DivisionLatency} = \left\lceil \begin{array}{c} \text{NoOverflow} \Rightarrow 3 + \left(\frac{34 - \text{divisorLength}}{4}\right) \\ \text{Overflow} \Rightarrow 2 \end{array} \right\rceil$$

   Where:

   $$\text{Overflow} = \left(\frac{x}{0}\right) \text{or} \left(\frac{\text{MaxNegativeNumber}}{-1}\right)$$

5. DivisionBlockage = DivisionLatency
6. Blocking the multiply instruction is dependent on the subsequent instruction. For any subsequent multiply instruction, the blockage is 1 clock and for any subsequent divide it is 2 clocks.
7. Assuming nonspeculative aligned access, on-chip memory, and available bus. For details, refer to **Section 6.6.5 Issuing Nonspeculative Load Instructions, Section 6.6.6 Executing Unaligned Instructions**, and **Section 6.6.9 Instruction Timing.**
8. Although a store (as well as **mtspr** for special registers external to the core) issued to the load/store unit buffer frees the core pipeline, the next load or store will not actually be performed on the bus until the bus is free.

8.2 INSTRUCTION EXECUTION TIMING EXAMPLES

All examples assume an instruction cache hit.

8.2.1 Data Cache Load

```
lwz   r12,64 (SP)
sub   r3,r12,3
addic r4,r14,1
mulli r5,r3,3
addi  r4,3(r0)
```

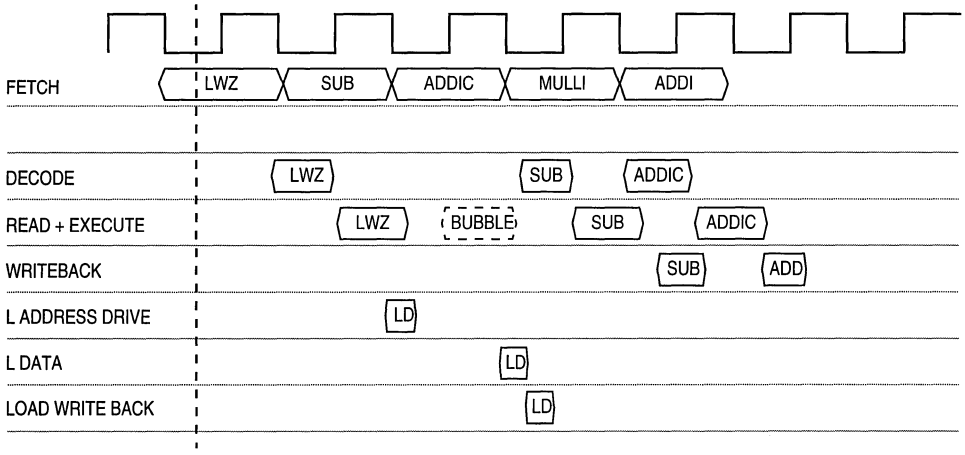


Figure 8-1. Example of a Data Cache Load

This is an example from a data cache with zero wait states. The **sub** instruction is dependent on the value loaded by the load to r12. This causes a bubble to occur in the instruction stream as shown in the execute line. Refer to **Section 8.2.2.2 Private Writeback Bus Load** for instances in which no such dependency exists.

8.2.2 Writeback

8.2.2.1 WRITEBACK ARBITRATION

mulli r12,r4,3
sub r3,r15,3
addic r4,r12,1

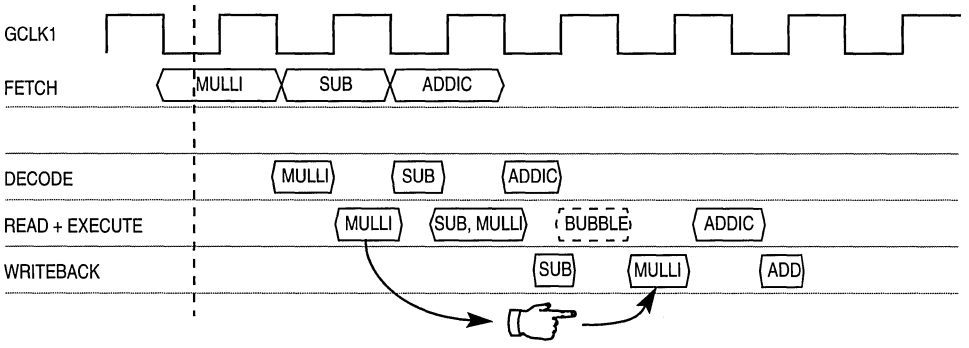


Figure 8-2. Example of a Writeback Arbitration

The **addic** instruction is dependent on the **mulli** result. Since the single-cycle instruction **sub** has priority on the writeback bus over the **mulli** and the **mulli** writeback is delayed one clock and causes a bubble in the execute stream.

mulli r12,r4,3
sub r3,r15,3
addic r4,r3,1

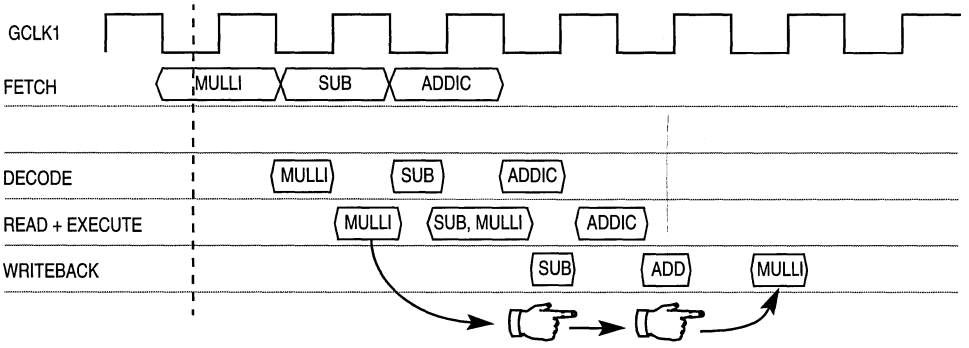


Figure 8-3. Another Example of a Writeback Arbitration

In this example, the **addic** instruction is dependent on the **sub** rather than on the **mulli**. Although the writeback of the **mulli** is delayed two clocks, there is no bubble in the execution stream.

8.2.2.2 PRIVATE WRITEBACK BUS LOAD

```
lwz   r12,64 (SP)
sub   r5,r5,3
cror  4,14,1
and   r3,r4.r5
xor   r4,r3,r5
or    r6,r12.r3
```

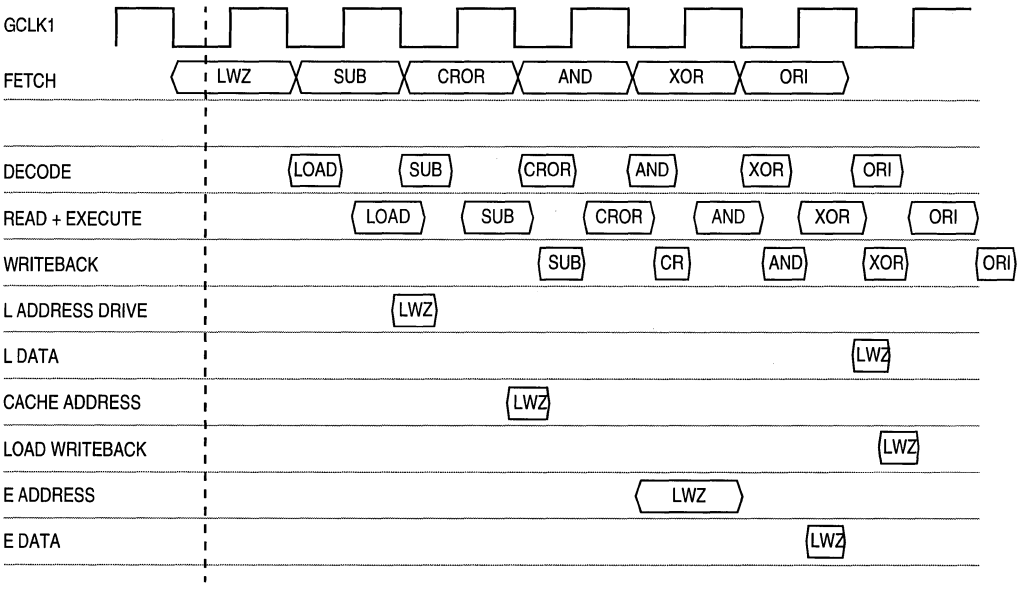



Figure 8-4. Example of a Private Writeback Bus Load

The load and the **xor** writeback in the same clock since they use the writeback bus in two different ticks.

8.2.3 Fastest External Load (Data Cache Miss)

```
lwz   r12,64 (SP)
sub   r3,r12,3
addic r4,r14,1
```

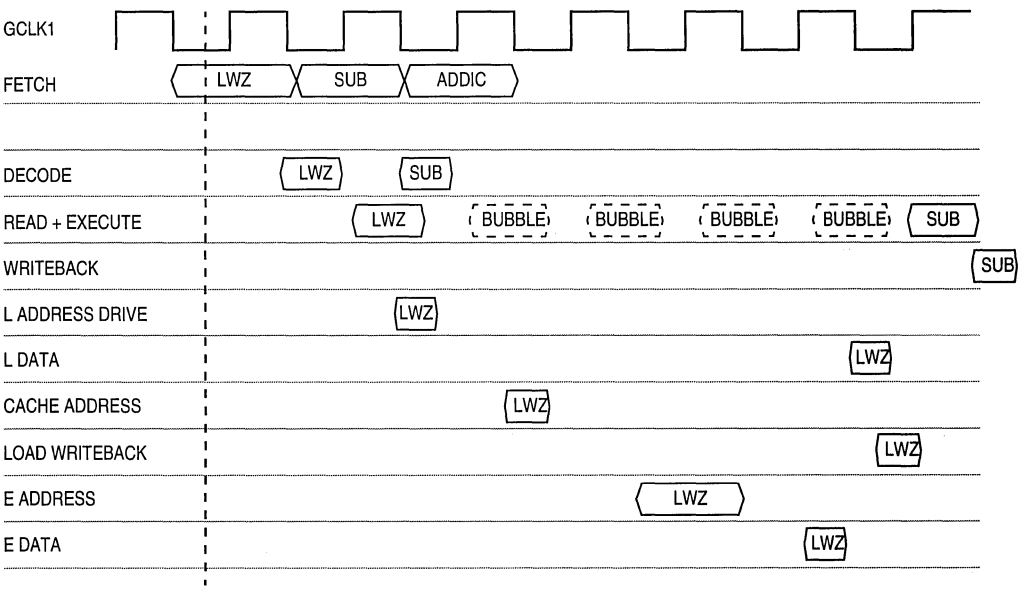


Figure 8-5. Example of an External Load

The **sub** instruction is dependent on the value read by the load. It causes three bubbles in the instruction execution stream. The external clock is shifted 90° relative to the internal clock.

8.2.4 A Full History Buffer

```
lwz   r12,64 (SP)
sub   r5,r5,3
addic r4,r14,1
and   r3,r4.r5
xor   r4,r3,r5
ori   r7,r8,1
```

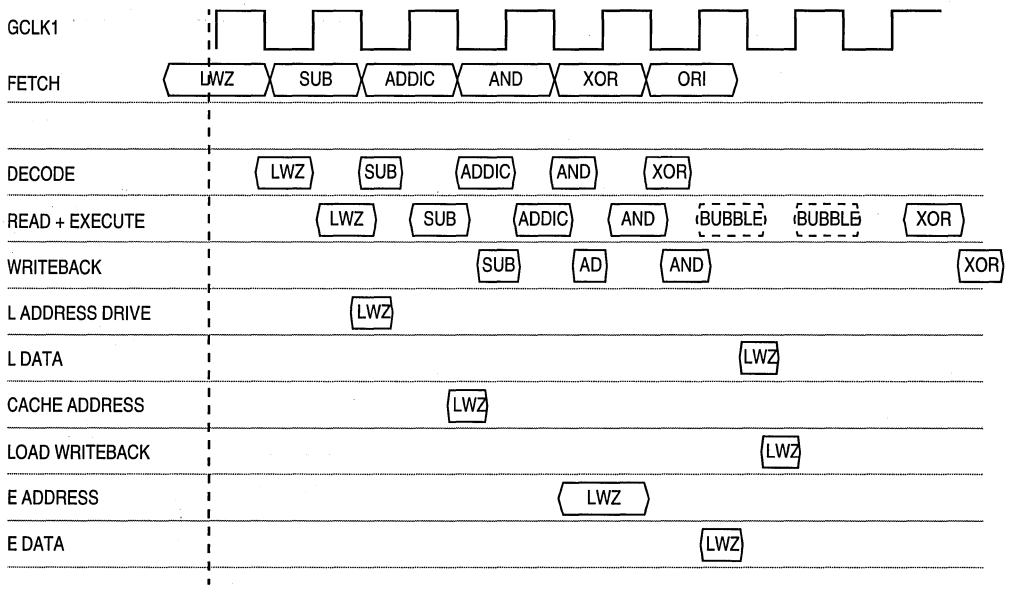


Figure 8-6. Example of a Full History Buffer

This example demonstrates the condition of a full history buffer. In this case, the history buffer is full from executing the **sub**, **add**, and **and** instructions. It takes one more bubble from the load writeback to allow further issue. This is the time for the history buffer to retire **sub**, **add**, and **and**.

8.2.5 Branch Folding

```
lwz   r12,64 (SP)
sub   r3,r12,3
addic r4,r14,1
bl    func
...
func:
mulli r5,r3,3
addi  r4,3(r0)
```

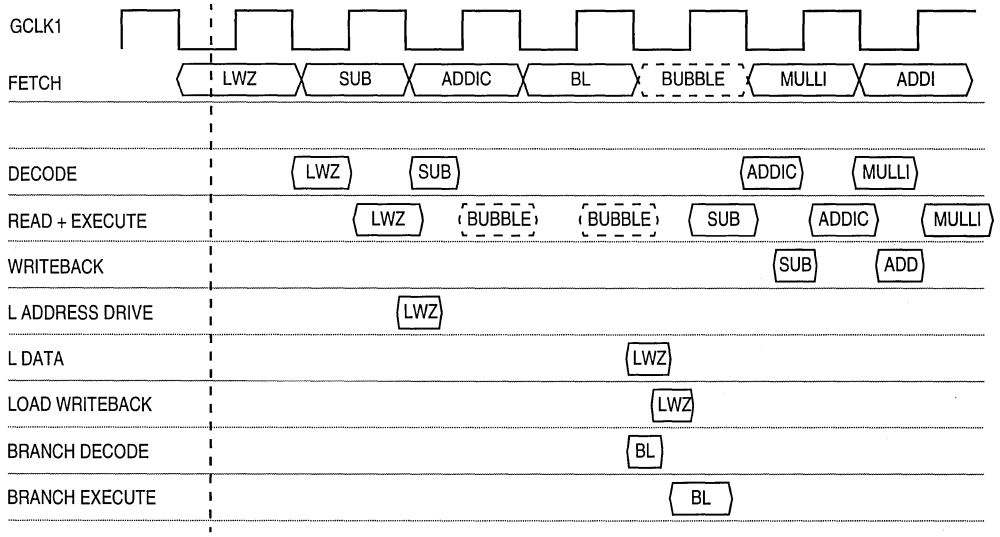


Figure 8-7. Example of Branch Folding

The **lwz** instruction accesses the internal storage with one wait state. The instruction prefetch queue and parallel operation of the branch unit allows the two bubbles caused by the **bl** issue and execution to overlap the two bubbles caused by the load. The issue of the branch itself is referred to as a bubble since no actual work is done by a branch.

8.2.6 Branch Prediction

while:
mulli r3,r12,r4
addi r4,3(r0)
...
lwz r12,64 (r2)
cmpi 0,r12,3
addic r6,r5,1
blt cr0,while
...

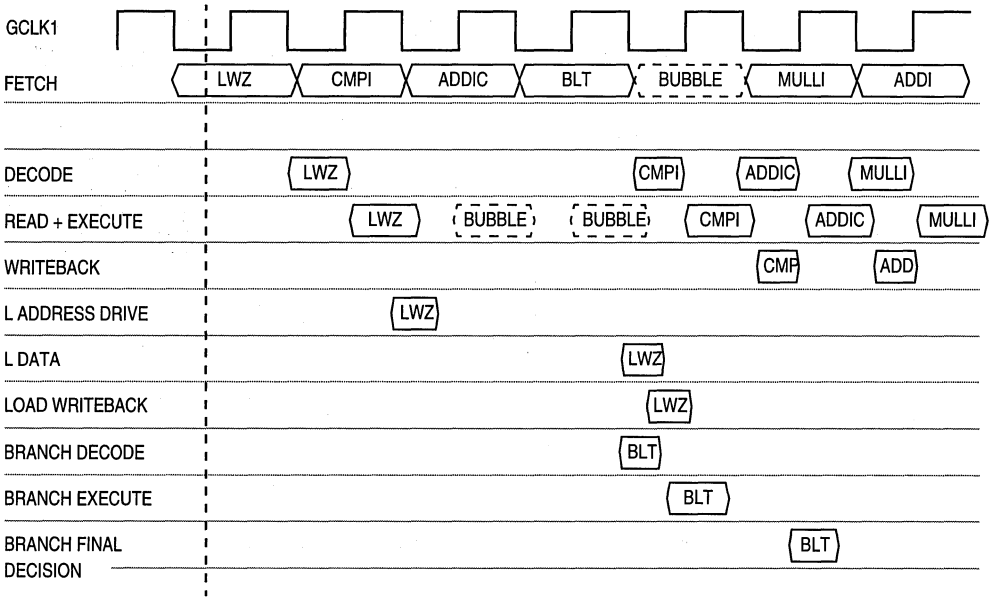


Figure 8-8. Example of Branch Prediction

In this example, the **blt** instruction is dependent on the **cmpi** instruction. Nevertheless, the branch unit predicts the correct path and allows an overlap of its bubbles with those of **lwz**. When the **cmpi** writes back, the branch unit reevaluates the decision and if correct prediction occurs no more action is taken and execution continues fluently. The fetched instructions on the predicted path are not allowed to execute before the condition is finally resolved. Instead, they are stacked in the instruction prefetch queue.

SECTION 9
INSTRUCTION CACHE

The MPC823 instruction cache is a 2K two-way, set associative storage area. It is organized into 64 sets, two lines per set and four words per line. Cache lines are aligned on 4-word boundaries in memory and can be used as an SRAM that allows the application to lock critical code segments that need fast and deterministic execution time. The cache access cycle begins with an instruction request from the instruction unit in the core. If a cache hit occurs, the instruction is delivered to the instruction unit and if a cache miss occurs, the cache initiates a burst read cycle on the internal bus with the address of the requested instruction. The first word received from the bus is considered the requested instruction. The cache forwards this instruction to the instruction unit of the core as soon as it is received from the internal bus. A cache line is then selected to receive the data that will be coming from the bus. A least recently used (LRU) replacement algorithm is used to select a line when no empty lines are available.

Each cache line can be used as an SRAM, thus allowing the application to lock critical code segments that need fast and deterministic execution time. Instruction cache coherency in a multiprocessor environment is maintained by the software and supported by a fast hardware invalidation capability. Figure 9-1 illustrates a block diagram view of the cache organization and Figure 9-2 illustrates a view of the cache's data path.

9.1 FEATURES

The following is a list of the instruction cache's main features:

- 2K Two-Way, Set-Associative at Four Words Per Line
- Implements the LRU Replacement Policy
- Parked On the Internal Bus
- Lockable Cache Lines
- "Critical word first", Burst Access
- Contains Stream Hit, Which Allows Fetching from the Burst Buffer and of the Word Currently on the Internal Bus
- Operates in Parallel with the Core to Maximize Performance
- Cache Control
  - ❑ Supports PowerPC™ invalidate instruction
  - ❑ Supports load and lock (cache line granularity)

- Supports Cache Inhibit
  - ❑ As a cache mode of operation (cache disable)
  - ❑ On memory regions (supported by the MMU)
- Efficiently Uses the Pipeline of the Internal Bus by Initiating a New Burst Cycle (If a Miss is Detected) While Bringing the Tail of the Previously Missed Line to the Cache.
- Performance Enhanced for Cache-Inhibited Regions by Fetching a Full Line to the Internal Burst Buffer. Instructions Stored in the Burst Buffer and Those Originated in a Cache-Inhibited Region Are Only Used Once Before Being Refetched.
- Instruction Unit Request Has Priority Over a Burst Buffer Write to an Array (Burst Buffer Holds Last Missed Data), thus Increasing the Overall Core Performance
- Miss Latency is Reduced by Sending Addresses to the Cache and Internal Bus Simultaneously and Aborting When a Hit Before a Cycle Goes External
- Minimum Operational Power Consumption
- Tags and Data Arrays Can be Accessed by the Core for Debugging and Testing Purposes
- Special Support is Available When the MPC823 Processor is in Debug Mode. Refer to **Section 9.9 Debug Support** for More Information.

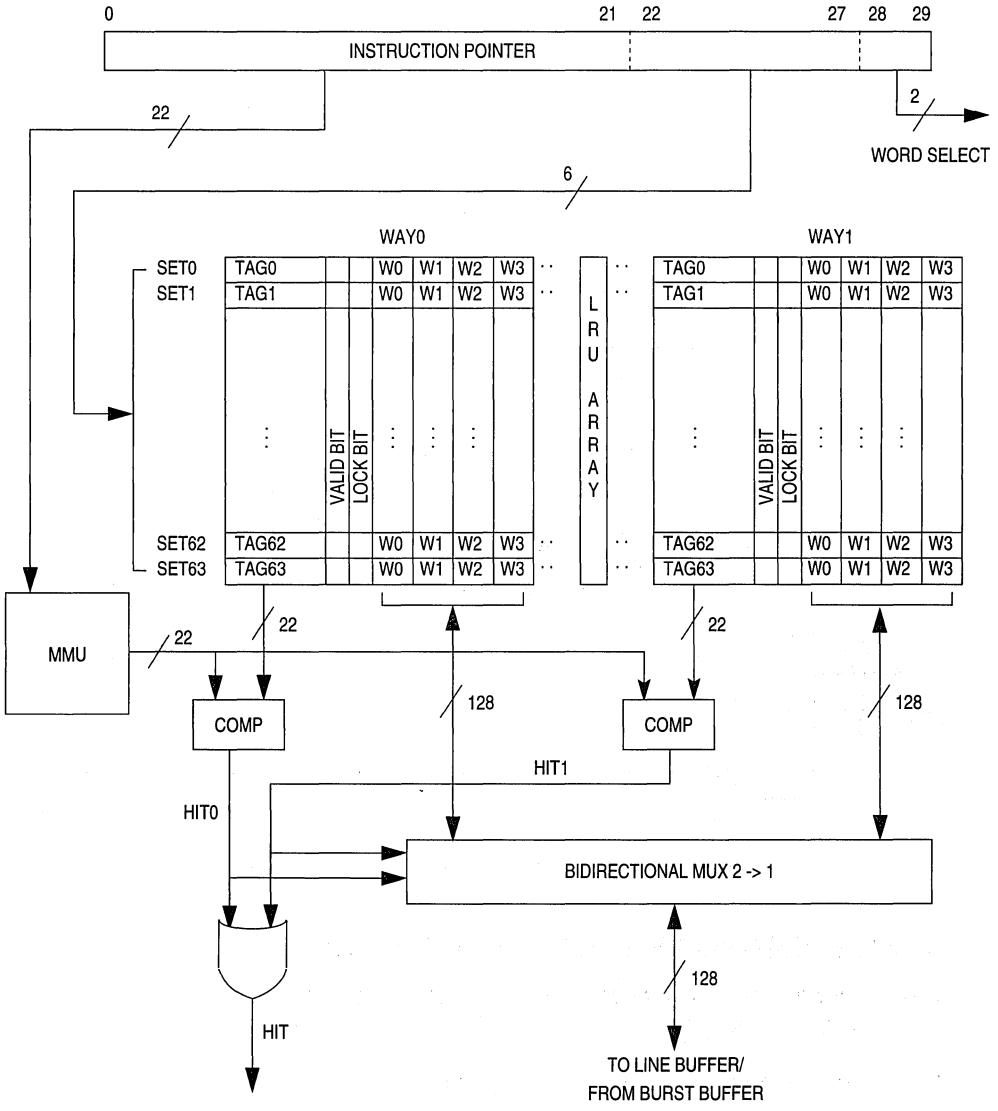


Figure 9-1. Instruction Cache Organization Block Diagram

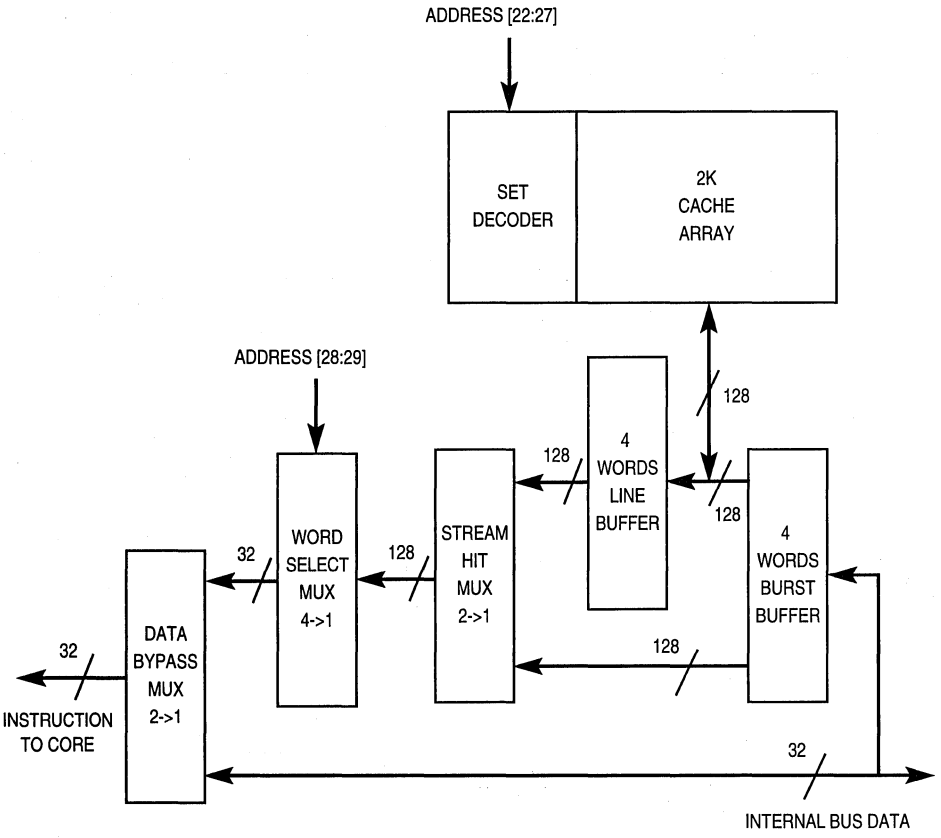


Figure 9-2. Cache Data Path Block Diagram

9.2 PROGRAMMING THE INSTRUCTION CACHE

Three special-purpose registers can be used to control the instruction cache with the **mfspr** and **mtspr** instructions:

- Instruction cache control and status register (IC_CSR)
- Instruction cache address register (IC_ADR)
- Instruction cache data port register (read-only) (IC_DAT)

These registers are privileged and any attempt to access them while the core is in the problem state (MSR_{PR}=1) results in a program interrupt.

9.2.1 Instruction Cache Control and Status Register

The instruction cache control and status register (IC_CSR) is used to configure and access the status of the instruction cache.

IC_CSR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	IEN	RESERVED			CMD			RESERVED			CCER1	CCER2	CCER3	RESERVED		
RESET	0	0			0			0			0	0	0	0		
R/W	R	—			R/W			—			R	R	R	—		
SPR	560															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED															
RESET	0															
R/W	—															
SPR	560															

NOTE: — = Undefined.

IEN—Instruction Cache Enable Status

This bit is read-only. Any attempt to write it is ignored. Enable or disable the instruction cache by writing to the CMD field.

0 = Instruction cache is disabled.
1 = Instruction cache is enabled.

Bits 1–3—Reserved

These bits are reserved and should be set to 0.

CMD—Command

The following commands can be written to the CMD field to control and configure the instruction cache. The machine must be in privilege mode (MSR_{PA}=1).

000 = Reserved.
001 = **CACHE ENABLE.**
010 = **CACHE DISABLE.**
011 = **LOAD & LOCK.**
100 = **UNLOCK LINE.**
101 = **UNLOCK ALL.**
110 = **INVALIDATE ALL.**
111 = Reserved.

Bits 7–9—Reserved

These bits are reserved and should be set to 0.

CCER1—Instruction Cache Error Type 1
This field is sticky and set by the hardware. It is read-only and cleared when read.

0 = No Error.
1 = Error.

CCER2—Instruction Cache Error Type 2
This field is sticky and set by the hardware. It is read-only and cleared when read.

0 = No Error.
1 = Error.

CCER3—Instruction Cache Error Type 3
This field is sticky and set by the hardware. It is read-only and cleared when read.

0 = No Error.
1 = Error.

Bits 13–31—Reserved
These bits are reserved and should be set to 0.

9.2.2 Instruction Cache Address Register

The instruction cache register (IC_ADR) contains addresses to be used in the command programmed in the IC_CSR.

IC_ADR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	ADR															
RESET	—															
R/W	R/W															
SPR	561															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	ADR															
RESET	—															
R/W	R/W															
SPR	561															

NOTE: — = Undefined.

ADR—Address
This field represents the address to be used in the command programmed in the IC_CSR. The format may vary depending on the selected cache operation.

9.2.3 Instruction Cache Data Port Register

The instruction cache data port register (IC_DAT) contains data that is received from the instruction cache.

IC_DAT

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	DAT															
RESET	—															
R/W	R/W															
SPR	562															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	DAT															
RESET	—															
R/W	R/W															
SPR	562															

NOTE: — = Undefined.

DAT—Data

This field represents the data received when reading information from the instruction cache. The format may vary depending on the selected cache

9.3 INSTRUCTION CACHE OPERATION

On an instruction fetch, bits 21-27 of the instruction's address point into the cache to retrieve the tags and data of one set. The tags from both ways are then compared against bits 0-20 of the instruction's address. If a match is found and the matched entry is valid, then it is a cache hit. If neither tags match or the matched tag is not valid, it is a cache miss. The instruction cache includes one burst buffer that holds the last line received from the bus and one line buffer that holds the last line retrieved from the cache array. If the requested data is found in one of these buffers, it can also be considered a cache hit. Refer to Figure 9-2 for more information. To minimize power consumption, the instruction cache attempts to make use of data stored in one of its internal buffers. Using a special indication from the core, it is possible to make sure that the requested data is in one of the buffers early enough that the cache array is not activated.

9.3.1 Instruction Cache Hit

When a cache hit occurs, bits 28-29 of the instruction address are used to select one word from the cache line whose tag matches the instruction pointer. The instruction is then immediately transferred to the instruction unit of the core.

9.3.2 Instruction Cache Miss

When an instruction cache miss occurs, the address of the missed instruction is driven on the internal bus with a 4-word burst transfer read request. A cache line is then selected to receive the data that will be coming from the bus. The selection algorithm gives first priority to invalid lines. If neither of the two lines in the selected set are invalid, then the least recently used line is selected for replacement. Locked lines are never replaced. The transfer begins with the word requested by the instruction unit, followed by the remaining words of the line, then by the word at the beginning of the lines.

When the missed instruction is received from the bus, it is immediately delivered to the instruction unit and also written to the burst buffer. As subsequent instructions are received from the bus, they are written into the burst buffer and delivered to the instruction unit either directly from the bus or from the burst buffer. When the line resides in the burst buffer, it is written to the cache array as long as it is not busy with an instruction unit request. If a bus error is encountered on the access to the requested instruction, then a machine check interrupt is taken. If a bus error occurs on any access to other words in the line, then the burst buffer is marked invalid and the line is not written to the array. However, if no bus error is encountered, the burst buffer is marked valid and eventually written to the array.

If you receive a cache-inhibit signal, the line is only written to the burst buffer and not to the cache. Instructions that are stored in the burst buffer and originate in a cache-inhibited memory region are only used once before they are refetched. Refer to **Section 9.4.6 Instruction Cache Read** for more information.

9.3.3 Instruction Fetch On A Predicted Path

The core allows branch prediction so branches can issue as early as possible. This mechanism allows instruction prefetch to continue while an unresolved branch is being computed and the condition is being evaluated. Instructions fetched after unresolved branches are considered fetched on a predicted path. These instructions may be discarded later if it turns out that the machine has followed the wrong path. To minimize power consumption, the MPC823 instruction cache does not initiate a miss sequence in most cases when the instruction is inside a predicted path. The MPC823 instruction cache evaluates fetch requests to see if they are inside a predicted path and if a hit is detected, the requested data is delivered to the core. However, if a cache miss is detected, the miss sequence is usually not initiated until the core finishes branch evaluation.

9.4 INSTRUCTION CACHE COMMANDS

The MPC823 instruction cache supports the PowerPC invalidate instruction with some additional commands that help control the cache and debug the information stored in it. The additional commands are implemented using the three special-purpose control registers mentioned previously in **Section 9.2 Programming the Instruction Cache**. Most of the commands are executed immediately after the control register is written and unable to generate any errors. Therefore, when executing these commands there is no need to check the error status in the IC_CSR.

Some commands may take some time to generate errors. In the current implementation, **LOAD & LOCK** is the only command to which this applies. Therefore, when executing these commands, you must insert an **isync** instruction immediately after the instruction cache command and check the error status in the IC_CSR after the **isync**. The error type bits in the IC_CSR are sticky, thus allowing you to perform a series of instruction cache commands before checking the termination status. These bits are set by the hardware and cleared by the software.

Only commands that are not immediately executed need to be followed by an **isync** instruction for the hardware to perform them correctly. However, all commands need to be followed by an **isync** to make sure all instruction fetches that are after the instruction cache commands in the program stream are affected by the instruction cache command. When the instruction cache is executing a command it is busy, so it stops any treatment of core requests. This eventually results in a machine stall.

9.4.1 Invalidating the Instruction Cache

The MPC823 implements the PowerPC instruction cache block invalidate (**icbi**) as it only pertains to the MPC823 instruction cache. This instruction does not broadcast on the external bus and the MPC823 does not snoop this instruction if it is broadcasted by other masters. This command is not privileged and has no associated error cases. The instruction cache performs this instruction in one clock cycle. To accurately calculate the latency of this instruction, bus latency should be taken into consideration.

The invalidate all instruction cache operation is privileged and any attempt to perform it when the core is in the problem state (MSR_{PR}=1) results in a program interrupt. When it is invoked and MSR_{PR} = 0, all valid lines in the cache, except the lines that are locked, are made invalid. As a result of this command, the lines' LRU points to an unlocked Way or to Way 0 if both lines are unlocked. This last feature is useful when initializing the instruction cache out of reset. For more information, refer to **Section 9.8 Reset Sequence**. To invalidate the whole cache, set the **INVALIDATE ALL** command in the IC_CSR. This command has no associated error cases. The instruction cache performs this instruction in one clock cycle. To accurately calculate the latency of this instruction, bus latency should be taken into consideration.

9.4.2 Loading and Locking the Instruction Cache

The **LOAD & LOCK** command is used to lock critical code segments in the instruction cache. This operation is privileged and any attempt to perform it when the core is in the problem state (MSR_{PR} = 1) results in a program interrupt. **LOAD & LOCK** is performed on a cache line granularity and after a line is locked, it operates as a regular instruction SRAM. It is not replaced during misses and it is not affected by invalidation commands. The hardware operation trusts the software to follow the exact steps mentioned in **Section 9.7 Updating Code And Memory Region Attributes**. To load and lock a line, follow these steps:

1. Read the error type bits in the IC_CSR to clear them.
2. Write the address of the line to be locked to the IC_ADR.
3. Set the **LOAD & LOCK** command in the IC_CSR.
4. Execute the **isync** instruction.
5. Return to Step 2 to load and lock more lines.
6. Read the error type bits in the IC_CSR to determine if the operation completed properly.

After the **LOAD & LOCK** command is written to the IC_CSR, the cache checks to see if the line containing the byte addressed by the IC_ADR is in the cache. If it is a hit, the line is locked and the command terminates with no exception. If it is not, a regular miss sequence is initiated. After the whole line is placed in the cache, the line is locked. You must check the error type bits in the IC_CSR to determine if the **LOAD & LOCK** operation completed properly. The **LOAD & LOCK** command can generate two errors:

- Type 1—A bus error occurs in one of the cycles that fetched the line.
- Type 2—There is no place to lock. It is your responsibility to make sure that there is at least one unlocked way in the appropriate set.

9.4.3 Unlocking A Line

The **UNLOCK LINE** command is used to unlock previously locked cache lines. This operation is privileged and any attempt to perform it when the core is in the problem state (MSR_{PR} = 1) results in a program interrupt. **UNLOCK LINE** is performed on a cache line granularity. If the line is found in the cache it is considered a hit, thus it is unlocked and operates as a regular valid cache line. If the line is not found in the cache it is considered a miss, there is no operation and the command terminates without an exception. To unlock a line, follow these steps:

1. Write the address of the line to be unlocked into the IC_ADR.
2. Set the **UNLOCK LINE** command in the IC_CSR.

This command has no error cases that you need to check. The instruction cache performs this instruction in one clock cycle. To accurately calculate the latency of this instruction, bus latency should be taken into consideration.

9.4.4 Unlocking the Entire Instruction Cache

The **UNLOCK ALL** command is used to unlock the entire instruction cache. This operation is privileged and any attempt to perform it when the core is in the problem state (MSR_{PR}=1) results in a program interrupt. It is performed on all cache lines. If a line is locked, it is unlocked and operates as a regular valid cache line. If a line is not locked or if it is invalid, no operation occurs. To unlock the whole cache, set the **UNLOCK ALL** command in the IC_CSR. This command has no associated error cases. The instruction cache performs this instruction in one clock cycle. To accurately calculate the latency of this instruction, bus latency should be taken into consideration.

9.4.5 Inhibiting the Instruction Cache

In the MPC823, there are two ways to inhibit the cache—using the MMU cache-inhibit attribute or the cache disable mode. To disable the instruction cache, set the **CACHE DISABLE** command in the IC_CSR. This operation is privileged and any attempt to perform it when the core is in the problem state (MSR_{PR}=1) results in a program interrupt. This command has no error cases that you need to check.

To enable the instruction cache, set the **CACHE ENABLE** command in the IC_CSR. This operation is privileged and any attempt to perform it when the core is in the problem state (MSR_{PR}=1) results in a program interrupt. This command has no error cases that you need to check. When fetching from cache-inhibited regions the full line is brought to the internal burst buffer. Instructions that originate in a cache-inhibited region and are stored in the burst buffer can be sent to the MPC823 core no more than once before being refetched. In the memory management unit, a memory region can be programmed as cache-inhibited. When changing a memory region to be cache inhibited, you must unlock all previously locked lines containing code that originated in this memory region, invalidate all lines containing code that originated in this memory region, and execute an **isync** instruction.

Note: Failure to follow these steps causes code from cache-inhibited regions to be left inside the cache and any reference to these regions will result in a cache hit. If a reference to a cache-inhibited region results in a cache hit, the data is sent to the core from the cache and not from memory.

When the MPC823 asserts the FRZ signal, it indicates that the MPC823 is under debug and all fetches from the cache are treated as if they were from the cache-inhibited memory region. For more information on cache debug support, refer to **Section 9.9 Debug Support**.

9.4.6 Instruction Cache Read

The MPC823 allows you to read all data stored in the instruction cache, including the content of the tags array. However, this operation is privileged and any attempt to perform it when the core is in the problem state (MSR_{PR} =1) results in a program interrupt. To read the data stored in the instruction cache, follow these steps:

1. Write the address of the data to be read to the IC_ADR. It is also possible to read this register for debugging purposes.
2. Read the IC_DAT register.

So that it can access all parts of the instruction cache, the IC_ADR is divided into the fields shown in the following table.

IC_ADR (CACHE READ COMMAND FORMAT)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
RESET	—															
R/W	R/W															
SPR	561															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED		TD	WAY	RESERVED		SET						WORD		RESERVED	
RESET	—		—	—	—		—						—		—	
R/W	R/W		R/W	R/W	R/W		R/W						R/W		R/W	
SPR	561															

NOTE: — = Undefined.

Bits 0–17—Reserved

These bits are reserved and should be set to 0.

TD—Tag or Data Select

0 = Select tag RAM.
1 = Select data RAM.

WAY—Way Select

0 = Select way 0 of cache array.
1 = Select way 1 of cache array.

Bits 20–21—Reserved

These bits are reserved and should be set to 0.

SET—Set Select

This field is used to select the set index of the cache array.

WORD—Word Select

This field is used to select a word within a set and way.

Bits 30–31—Reserved

These bits are reserved and should be set to 0.

When read from the data array, the 32 bits of the word selected by the IC_ADR is placed in the targeted general-purpose register. When read from the tag array, the 22 bits of the tag and related information that is selected by the IC_ADR are all placed in the targeted general-purpose register. The following table provides the bit layout of the instruction cache data register when reading a tag.

IC_DAT (TAG READ FORMAT)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	TAG															
RESET	—															
R/W	R/W															
SPR	562															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	TAG						V	L	LRU	RESERVED						
RESET	—						—	—	—	—						
R/W	R/W						R/W	R/W	R	R/W						
SPR	562															

NOTE: — = Undefined.

TAG—Tag Select

This field contains the upper 22 bits of the address.

V—Valid Entry

0 = Entry is not valid.
1 = Entry is valid.

L—Lock Entry

0 = Entry is unlocked.
1 = Entry is locked.

LRU—Least-Recently Used

This bit indicates that the entry is aged or least recently used.

Bits 25–31—Reserved

These bits are reserved and should be set to 0.

9.4.7 Instruction Cache Write

Instruction cache write is only enabled when the MPC823 is in test mode.

9.5 RESTRICTIONS

Zero wait state devices that are placed on the internal bus are considered to be in the cache-inhibited memory region and the hardware correct operation trusts the software to follow the exact steps mentioned in **Section 9.7 Updating Code And Memory Region Attributes**. It is not recommended that you perform **LOAD & LOCK** from zero wait state devices that are placed on the internal bus, especially since it is not guaranteed that the data will be fetched from the instruction cache. In most cases, it is fetched from the device, but found in the instruction cache.

9.6 INSTRUCTION CACHE COHERENCY

Cache coherency in a multiprocessor environment is maintained by the software and supported by the invalidation mechanism as described above. All instruction storage is considered to be coherent, not required, mode.

9.7 UPDATING CODE AND MEMORY REGION ATTRIBUTES

To update the code or change the programming of the memory regions in the chip-select logic, follow these steps:

1. Update the code and change the memory region programming in the chip-select logic.
2. Execute the **sync** instruction to ensure that the update/change operation has finished.
3. Unlock all locked lines that contain code that was updated.
4. Invalidate all lines that contain code that was updated.
5. Execute the **isync** instruction.

9.8 RESET SEQUENCE

To simplify the debug task of the system, the instruction cache is only disabled during hardware reset ($IC\_CSR_{EN} = 0$). This feature enables you to investigate the exact state of the instruction cache prior to the event that asserts the reset. To ensure proper operation of the instruction cache after reset, the **UNLOCK ALL**, **INVALIDATE ALL**, and **INSTRUCTION CACHE ENABLE** commands must be executed.

9.9 DEBUG SUPPORT

The MPC823 can be debugged either in debug mode or by a software monitor debugger. In both cases, the core asserts the internal FRZ signal. When FRZ is asserted the instruction cache treats all misses as if they were from cache-inhibited regions and, assuming the debug routine is not in the instruction cache, the cache state remains exactly the same. When FRZ is asserted, hits are still read from the array and the LRU bits are updated. Therefore, in the simple case of the debug routine it is read from memory like any other miss. However, for performance reasons, it might be preferable to run the debug routine from the cache. Follow these steps with little variation:

1. Save both ways of the sets that are needed for the debug routine by reading the tag, LRU bit, valid bit, and lock bit values.
2. Unlock the locked ways in the selected sets.
3. Use a **LOAD & LOCK** command to load and lock the debug routine into the instruction cache (**LOAD & LOCK** operates the same when FRZ is asserted).
4. Run the debug routine. All accesses to it will result in hits.

After the debug routine has completed, the old state of the instruction cache can be restored by following these steps:

1. Unlock and invalidate all the sets that are used by the debug routine (both ways).
2. Use a **LOAD & LOCK** command to restore the old sets.
3. Unlock the ways that were not previously locked.
4. To restore the old state of the LRU, make sure the last access is made in the MRU way. An access in this description is either **LOAD & LOCK** or **UNLOCK LINE**.

9.9.1 Fetching Instructions From The Development Port

When the MPC823 is in debug mode all instructions are fetched from the development port, regardless of the address generated by the core. Therefore, the instruction cache is practically bypassed when the MPC823 is in debug mode.

SECTION 10
DATA CACHE

The MPC823 data cache is a 1K two-way, set-associative cache. It is organized into 32 sets, two lines per set and four words per line. Cache lines are aligned on 4-word boundaries in memory and can be used as an SRAM that allows the application to lock critical data segments that need a fast and deterministic execution time. Two state bits are included in each cache line and implement invalid, modified-valid, and unmodified-valid states of the data cache. Cache coherency in a multiprocessor environment is maintained by the software and supported by a fast hardware invalidation capability. The cache is designed for both writeback and writethrough modes of operation and a least recently used (LRU) replacement algorithm is used to select a line when no empty lines are available.

10.1 FEATURES

The following is a list of the data cache's main features:

- 1K Two-Way, Set Associative, and Physically Addressed
- Single-Cycle Cache Access On Hit and 1 Clock Latency Added for Miss
- Four Word Line Size
- "Critical Word First" and Four Word Burst Line Fill
- Implements LRU Replacement Policy
- 32-Bit Interface to Load/Store Unit
- One-Word Write Buffer
- Lockable Cache Line Granularity
- Copyback/Writethrough Operation is Programmed Per Memory Management Unit Page
- Coherency is Only Maintained By the Software and No Bus Snooping is Supported
- Cache Operation is Blocked Under Miss, Until the Critical Word is Delivered to the Core
- Hit Under Miss Operation
- Full Data Cache PowerPC™ Control Operations
- Implementation-Specific Single Operation

10.2 ORGANIZATION OF THE DATA CACHE

The data cache is a 1K two-way, set associative, physically addressed cache that has 16-byte line and a 32-bit data path to and from the load/store unit, which allows for a 4-byte transfer per cycle.

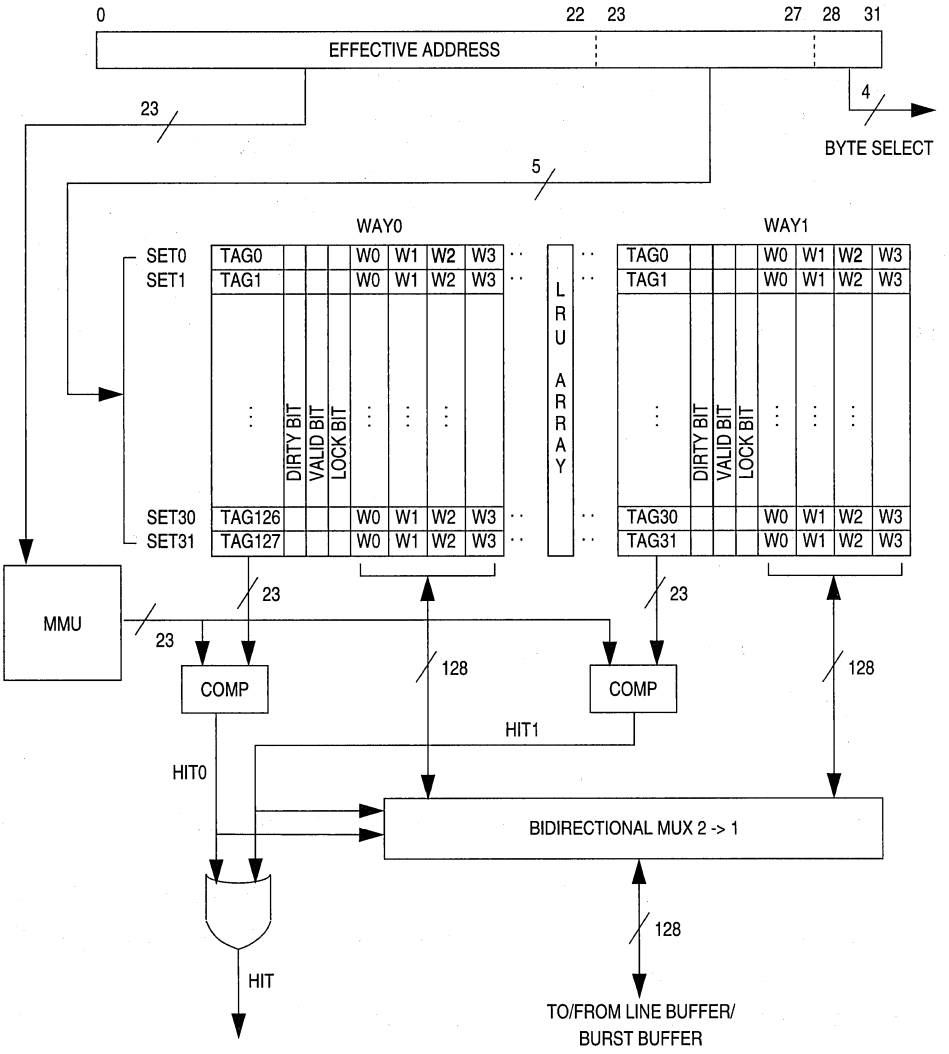


Figure 10-1. Data Cache Organization

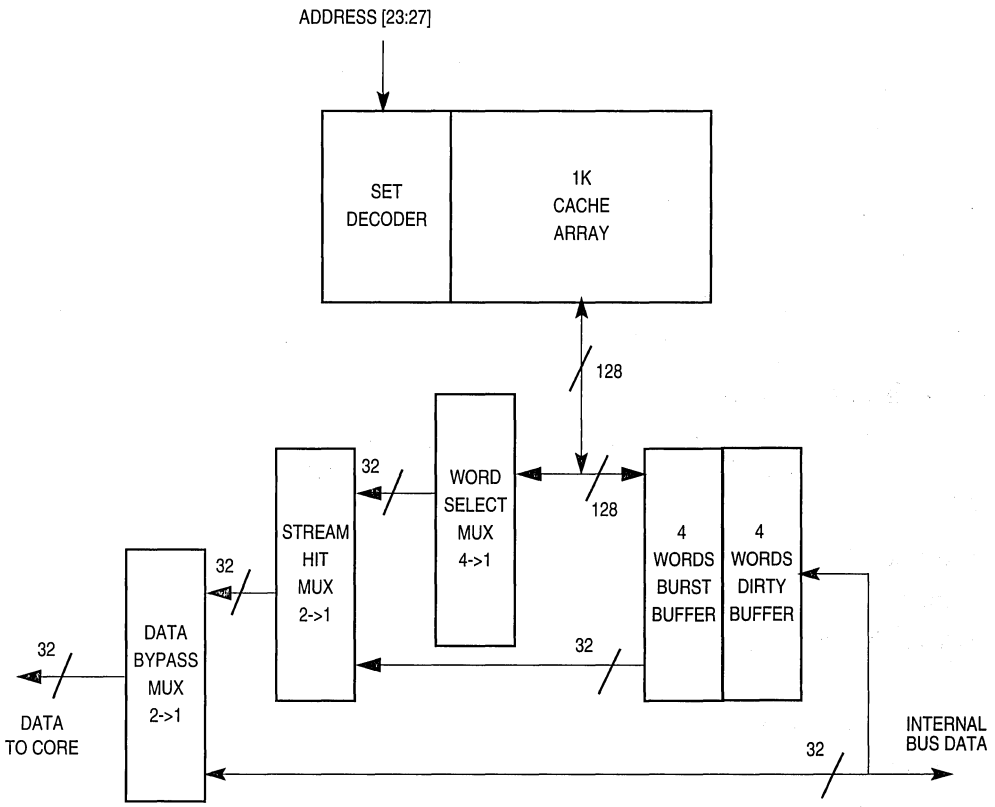


Figure 10-2. Cache Data Path Block Diagram

10.3 PROGRAMMING THE DATA CACHE

10.3.1 PowerPC Architecture Instructions

The following PowerPC instructions are supported by the data cache.

10.3.1.1 PowerPC USER INSTRUCTION SET ARCHITECTURE (BOOK I)

The data cache supports the **sync** instruction through a cache pipe clean indication to the core.

10.3.1.2 PowerPC VIRTUAL ENVIRONMENT ARCHITECTURE (BOOK II)

The data cache supports the following instructions:

- Data cache block flush (**dcbf**)
- Data cache block store (**dcbst**)
- Data cache block touch (**dcbt**)
- Data cache block touch for store (**dcbtst**)
- Data cache block set to zero (**dcbz**)

10.3.1.3 PowerPC OPERATING ENVIRONMENT ARCHITECTURE (BOOK III). The data cache supports the **dcbi** (data cache block invalidate) instruction.

10.3.2 Implementation-Specific Operations

The MPC823 data cache includes some extended features in addition to those in the PowerPC architecture. The following are implementation-specific operations supported by the MPC823 data cache:

- Block lock
- Block unlock
- Invalidate all
- Unlock all
- Flush cache line
- Read tags
- Read registers

10.3.3 Special Registers of the Data Cache

The PowerPC special registers are accessed via the **mtspr** and **mfspr** instructions. The following registers are used to control the data cache:

- Data cache control and status register (DC_CSR)
- Data cache address register (DC_ADR)
- Data cache data register (DC_DAT)

These registers are privileged and any attempt to access them while the core is in the problem state (MSR_{PR} =1) results in a program interrupt.

10.3.3.1 DATA CACHE CONTROL AND STATUS REGISTER. The data cache control and status register (DC_CSR) is used to configure and access the status of the data cache.

DC_CSR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	DEN	DFWT	LES	RES	CMD				RESERVED		CCER1	CCER2	CCER3	RESERVED		
RESET	0	0	0	0	0				0		0	0	0	0		
R/W	R	R	R	R/W	R/W				R/W		R	R	R	R/W		
SPR	568															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED															
RESET	0															
R/W	R/W															
SPR	568															

NOTE: — = Undefined.

DEN—Data Cache Enable Status

This bit is read-only and any attempt to write to it is ignored. Write to the CMD field to enable or disable the data cache.

0 = Data cache is disabled.
1 = Data cache is enabled.

DFWT—Data Cache Force Writethrough

This bit is read-only and any attempt to write to it is ignored. Write to the CMD field to set or force writethrough mode.

0 = Data cache mode determined by memory management unit.
1 = Data cache is forced writethrough.

LES—Little-Endian Swap

Refer to **Section 14 Endian Modes** for details about how this bit is used to achieve the required endian behavior. This bit is read-only. Write to the CMD field to set or clear little-endian swap mode.

0 = Address of the data and the instruction caches is the unchanged address from the core. No byte swap is done on the data and instruction caches' external accesses.
1 = Address munging performed by the core is reversed before accessing the data cache, the instruction cache and storage. Byte swap is performed for the instruction and data caches' external accesses. This bit is a read-only bit and any attempt to write to it is ignored.

Bits 3, 8, and 9—Reserved

These bits are reserved and should be set to 0.

CMD—Command

The following commands can be written to the CMD field to control and configure the data cache. The machine must be in privilege mode (MSR_{PR}=1).

0000 = Reserved.
0010 = **DATA CACHE ENABLE.**
0100 = **DATA CACHE DISABLE.**
0110 = **LOCK LINE.**
1000 = **UNLOCK LINE.**
1010 = **UNLOCK ALL.**
1100 = **INVALIDATE ALL.**
1110 = **FLUSH DATA CACHE LINE.**
0001 = **SET FORCE WRITETHROUGH MODE.**
0011 = **CLEAR FORCE WRITETHROUGH MODE.**
0101 = **SET LITTLE-ENDIAN SWAP MODE.**
0111 = **CLEAR LITTLE-ENDIAN SWAP MODE.**
Others = Reserved.

CCER1—Data Cache Error Type 1

This field is sticky and set by the hardware. It is read-only and cleared when read.

0 = No Error.
1 = Error.

CCER2—Data Cache Error Type 2

This field is sticky and set by the hardware. It is read-only and cleared when read.

0 = No Error.
1 = Error.

CCER3—Data Cache Error Type 3

This field is sticky and set by the hardware. It is read-only and cleared when read.

0 = No Error.
1 = Error.

Bits 13–31—Reserved

These bits are reserved and should be set to 0.

10.3.3.2 DATA CACHE ADDRESS REGISTER. The data cache address register (DC_ADR) contains the address to be used in the command programmed in the CMD field of the DC_CSR. It also may contain the internal address for the read tags and register operation.

DC_ADR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	ADR															
RESET	—															
R/W	R/W															
SPR	569															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	ADR															
RESET	—															
R/W	R/W															
SPR	569															

NOTE: — = Undefined.

ADR—Address

This field represents the address to be used in the command programmed in the DC_CSR. It is also the internal address used to read tags and registers.

10.3.3.3 READING THE CACHE STRUCTURES. To read the data stored in the data cache tags or registers, follow these steps:

1. Write to the DC_ADR. This register can also be read for debugging purposes.
2. Read the DC_DAT register.

The DC_ADR must be configured into the following fields before the internal parts of the data cache are read from the DC_DAT register.

DC_ADR (CACHE READ COMMAND FORMAT)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
RESET	—															
R/W	R/W															
SPR	569															

BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED		RT	WAY	RESERVED			WAY					WORD		RESERVED	
RESET	—		—	—	—			—					—		—	
R/W	R/W		R/W	R/W	R/W			R/W					R/W		R/W	
SPR	569															

NOTE: — = Undefined.

Bits 0–17—Reserved
These bits are reserved and should be set to 0.

RT—Register or Tag Selection
0 = Select tag operation.
1 = Select register operation.

WAY—Way Selection
0 = Select Way 0 of the cache array.
1 = Select Way 1 of the cache array.

Bits 20–22—Reserved
This bit is reserved and should be set to 0.

SET—Set Selection
This field is used to select the index of the cache array.

WORD—Word Selection
This field is used to select a word within a set and way.

Bits 30–31—Reserved
These bits are reserved and should be set to 0.

The register number field specifies the register to be read. The following registers and their encoding are supported:

- 0x00—Copyback data register 0
- 0x01—Copyback data register 1
- 0x02—Copyback data register 2
- 0x03—Copyback data register 3
- 0x04—Copyback address register

When reading from the DC_DAT register, the 23 bits of the tag and related information that is selected by the DC_ADR are placed in the targeted general-purpose register. The following table illustrates the DC_DAT register's bit layout when reading a tag. Writing to DC_DAT is illegal. A write to DC_DAT results in an undefined data cache state.

DC_DAT (TAG READ FORMAT)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	TAG															
RESET	—															
R/W	R/W															
SPR	570															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	TAG							L	LRU	D	V	RESERVED				
RESET	—							—	—	—	—	—				
R/W	R/W							R/W	R/W	R/W	R/W	R/W				
SPR	570															

NOTE: — = Undefined.

TAG—Tag Selection

This field contains the upper 23 bits of the address.

L—Lock Entry

0 = Cache entry is unlocked.
1 = Cache entry is locked.

LRU—Least Recently Used

0 = This entry is not aged or least-recently used.
1 = This entry is aged or least-recently used.

D—Dirty or Clean Cache Line

0 = This entry has not been modified since it was read from memory.
1 = This entry has been modified since it was read from memory.

V—Valid Cache Line

0 = Entry is not valid.
1 = Entry is valid.

Bits 27–31—Reserved

These bits are reserved and should be set to 0.

10.4 OPERATING THE DATA CACHE

The data cache is a three-state design. Two bits are included in each cache line to maintain the line's state information. The status bits keep track of whether or not the line is valid or if it has been modified relative to memory. These modes are invalid, modified-valid, and unmodified-valid.

10.4.1 Data Cache Read

There are two possible outcomes of a data cache read:

- Read Hit—On a cache hit, the requested word is immediately transferred to the load/store unit and the LRU state of the set is updated. No state transition occurs and the access time is 1 clock (zero wait state).
- Read Miss—A line in the cache is selected to hold the data that will be fetched from memory. The selection algorithm gives first priority to invalid lines and if both lines are invalid, the way zero line is selected first. If neither of the two candidate lines in the selected set is invalid, then one of the lines is selected by the LRU algorithm for replacement. If the selected line is valid-modified (dirty), then it is kept in a special buffer to be written out (flushed) to memory or a later time.

  Subsequently, the address of the missed entry is sent to the system interface unit with a request to retrieve the cache line. The system interface unit arbitrates for the bus and initiates a 4-word burst transfer read request. The transfer begins with the aligned word containing the missed data, followed by the remaining word in the line, then by the word at the beginning of the line (wraparound). As the missed word is received from the bus, it is delivered (forwarded) directly to the load/store unit. When the line has been fully received, it is written into the cache. The data cache supports further requests as long as they hit in the cache immediately after the arrival of the critical word.

  After the line with the requested data has been brought from memory, the dirty line kept in the buffer is sent to the system interface unit to be written out (flushed) to memory. If a bus error is detected during the fetch of the missed "critical word", a machine check interrupt is generated. If a bus error occurs on any other word in the line transfer, the line is marked invalid. On the other hand, if no bus error is encountered, the cache line is marked unmodified-valid. If a bus error is detected during the dirty line flush, a machine check interrupt is generated (the dirty line flush error is an imprecise interrupt). For more information about reading the address and data of a line, see **Section 10.3.3.3 Reading the Cache Structures**.

10.4.2 Data Cache Write

The cache operates in either writethrough or copyback mode, depending on how the memory management unit is programmed. If two logical blocks map to the same physical block, it is considered a programming error for them to specify different cache write policies.

10.4.2.1 COPYBACK MODE

In copyback mode, write operations do not necessarily update the external memory. For this reason, copyback mode is the preferred mode of operation when it is necessary to keep bus bandwidth usage and power consumption at a minimum. The possible outcomes of a data cache write in copyback mode are:

- Write Hit to Modified Line—Data is simply written into the cache with no state transition. The LRU of the set is updated to point to the way holding the hit data.
- Write Hit to Unmodified Line—Data is written into the cache and the line is marked modified. The LRU of the set is updated to point to the way holding the hit data.
- Write Miss—A line in the cache is selected to hold the data that is fetched from memory. The selection algorithm gives first priority to invalid lines if both lines are invalid the way zero line is selected first. If neither of the two candidate lines in the selected set is invalid, then one of the lines is selected by the LRU algorithm for replacement. If the selected line is valid-modified (dirty), it is kept in a special buffer to be written out (flushed) to memory at a later time.

  Subsequently, the address of the missed entry is sent to the system interface unit with a request to retrieve the cache line. The system interface unit arbitrates for the bus and initiates a 4-word burst transfer read request. The transfer begins with the aligned word containing the missed data (the critical word first), followed by the remaining word in the line, then by the word at the beginning of the line (wraparound). As the missed word is received from the bus, it is merged with the data to be written. When the line has been fully received, it is written into the cache. Once the line fill is complete, the new store data is written into the cache and the line is marked modified-valid (dirty). At this point, if the machine stalls while waiting for the store to complete, execution is allowed to resume. The data cache does not support further requests until after the whole line arrives.

  After the line with the requested data has been brought from memory, the dirty line kept in the buffer is sent to the system interface unit to be written out (flushed) to memory. The data cache can support further requests as long as they hit in the cache while flushing the dirty line to memory. If a bus error is detected during a fetch of the missed line (even on a word not accessed by the load/store unit) the cache line is not modified and a machine check interrupt is generated. If a bus error is detected during the dirty line flush, a machine check interrupt is generated (the dirty line flush error is an imprecise interrupt). For more information about reading the address and data of a line, see **Section 10.3.1.2 PowerPC Virtual Environment Architecture (Book II)**.

10.4.2.2 WRITETHROUGH MODE. In writethrough mode, store operations always update memory. This mode is used when external memory and internal cache images must agree. It gives a lower worst-case interrupt latency at the expense of average performance (for example, if it does not have to do flush accesses). The possible outcomes of a data cache write in writethrough mode are:

- Write Hit—Data is written into both the cache and memory, but the cache state is not changed. The LRU of the set is updated to point to the way holding the hit data. If a bus error is detected during the write cycle, the cache is still updated and a machine check interrupt is generated.
- Write Miss—Data is only written into memory, not to the cache (write no allocate) and no state transition occurs. The LRU is not changed, but if a bus error is detected during the write cycle, a machine check interrupt is generated.

10.4.3 Data Cache Inhibited Accesses

If the cache access is to a page that has the CI bit set in the memory management unit, one of the following outcomes will occur:

- Hit to Modified or Unmodified Line—This is considered a programming error if the targeted location copy of a **load**, **store**, or **dcbz** instruction to cache inhibit storage is in the cache. The result is boundedly undefined.
- Read Miss—Data is read from memory, but not placed in the cache. The cache's status is unaffected.
- Write Miss—Data is written through to memory, but not placed in the cache. The cache's status is unaffected.

10.4.4 Data Cache Freeze

The MPC823 can be debugged either in debug mode or by a software monitor debugger. The data cache is frozen when the FRZ signal is asserted so that the data cache can be examined for debugging purposes. For a detailed description of MPC823 debug support refer to **Section 20 Development Capabilities and Interface**. When FRZ is asserted, the possible outcomes are:

- Read Miss—Data is read from memory, but not placed in the cache. The cache's status is unaffected.
- Read Hit—Data is read from the cache, but LRU is not updated.
- Write Miss/Hit—Data cache operates in writethrough mode, but LRU is not updated.
- **dcbz** Instruction Miss/Hit—Data is written into cache and memory, but LRU is not updated.
- **dcbst/dcbf/dcbi** Instructions—The data cache and memory is updated according to the PowerPC architecture, but LRU is not updated.

10.4.5 Data Cache Coherency

The MPC823 data cache provides no support for snooping external bus activity. All coherency between the internal caches and memory/devices external to the extended core must be controlled by the software. In addition, there is no mechanism provided for DMA or other internal masters to access the data cache directly.

10.5 DATA CACHE COMMANDS

10.5.1 Flushing and Invalidating the Cache

The MPC823 allows the software to control how the data cache is flushed and invalidated. The data cache can be invalidated by writing the **UNLOCK ALL** and **INVALIDATE ALL** commands to the DC_CSR. The data cache is not automatically invalidated on reset. It must be invalidated under software control. The data cache can be flushed by a software loop using the **dcbst** or **dcbf** instructions or the implementation-specific **CACHE LINE FLUSH** command. Notice that the PowerPC architecture instructions flush a line indexed by the effective address, while the implementation-specific command indexes a line by its physical set index within the data cache.

When flushing must be restricted to a specific memory area or the architecture must be compliant, it is recommended that you use the PowerPC architecture instructions. However, if the entire data cache must be flushed and there is no concern for compatibility, the implementation-specific command is more efficient. If a bus error occurs while executing the **dcbf** and **dcbst** instructions or the implementation-specific **CACHE LINE FLUSH** command, the data of the cache line specified by these operations must be retrieved from the copyback data registers rather than from the data cache array.

10.5.2 Enabling and Disabling the Cache

The data cache can be enabled or disabled by writing the **DATA CACHE ENABLE** and **DATA CACHE DISABLE** commands to the DC_CSR. In the disabled state, the cache tag state bits are ignored and all accesses are propagated to the bus as single beat transactions. The default after the reset state of the data cache is disabled. Disabling the data cache does not affect the data address translation logic and translation is still controlled by the MSR_{DR} bit. Any write to the DC_CSR must be preceded by a **sync** instruction. This prevents the data cache from being disabled or enabled in the middle of a data access. When the data cache generates an interrupt as a result of a bus error on the **COPYBACK** or implementation-specific **CACHE LINE FLUSH** command, it enters the disable state. Operation of the cache when it is disabled is similar to cache-inhibit operation.

10.5.3 Locking and Unlocking the Cache

Each line of the data cache can be independently locked by writing the **LOCK LINE** command to the DC_CSR. Replacement line fills are not performed to a locked line. A flush or invalidation of a locked line cache is ignored by the data cache. Any write to the DC_CSR must be preceded by a **sync** instruction, which prevents a cache from being locked during a line fill. Use the **UNLOCK LINE** or **UNLOCK ALL** commands to unlock the cache.

10.5.4 Data Cache Instructions

10.5.4.1 dcbi, dcbst, dcbf AND dcbz INSTRUCTIONS

The **dcbz**, **dcbi**, **dcbst**, and **dcbf** instructions operate on a block basis of cache line, which is 16 bytes (4 words) long. A data TLB miss exception is generated if the effective address of one of these instructions cannot be translated and data address relocation is enabled.

10.5.4.2 TOUCH. The **dcbt** and **dcbtst** instructions of the MPC823 operate on a block basis of cache line, which is 16 bytes (4 words) long. Touch instructions initiate bus transfers to bring in a cache line of data from memory. They become no operation instructions if the effective address cannot be translated when the memory management unit is enabled.

10.5.4.3 STORAGE SYNCHRONIZATION/RESERVATION. The MPC823 does not support the multimaster storage protocol.

10.5.5 Data Cache Read

To allow debug and recovery actions, the MPC823 allows the content of the tags array as well as the last copyback address and data buffers to be read. See **Section 10.3.3 Special Registers of the Data Cache** for details. This operation is privileged and any attempt to perform it when the core is in the problem state (MSR_{PR} =1) results in a program interrupt.

SECTION 11
MEMORY MANAGEMENT UNIT

The MPC823 implements a virtual memory management scheme that provides cache control, storage access protection, and effective-to-real address translation. This implementation includes separate instruction and data memory management units. The MPC823 memory management unit is compliant with the *PowerPC Microprocessor Family: The Programming Environment for 32-Bit Microprocessors* manual in relation to the supported attributes, except that two new modes of operation have been added:

- PowerPC™ mode with extended encoding
- Domain manager mode

Available protection granularity sizes are page (4K, 16K, 512K, or 8M) or 1K subpage (1K subpage resolution is supported for 4K pages only). Hereafter, the prefix Mx_ appears before a memory management unit control register name that corresponds to both the instruction and data cache conditions.

11.1 FEATURES

The following is a list of the memory management unit's main features:

- 8-Entry Fully Associative Data and Instruction Translation Lookaside Buffers (TLBs)
- Multiple Page Size Support
- High Performance
- Supports Maximum of 16 Virtual Address Spaces
- Supports 16 Access Protection Groups
- Each Entry Can Be Programmed to Match Problem Accesses, Privileged Accesses, or Both
- PowerPC MSR_{IR} and MSR_{DR} Controls Memory Management Unit Translation and Protection
- Supports PowerPC **tlbie** and **tlbia** Instructions. No **tlbsync** Instruction Is Supported, But It Is Implemented As a NOP Instruction.
- PowerPC **mtspr/mfspr** Instructions To and From the Implementation-Specific Special-Purpose Registers Can be Used for Programming
- A Scratch Register Is Available for Software Tablewalks
- Designed for Minimum Power Consumption

11.2 ADDRESS TRANSLATION

The MPC823 core generates 32-bit effective addresses and, when enabled, the memory management unit translates the effective address to a real address that is used for cache or memory access. If disabled, the effective address is passed as the real address to the memory, which bypasses the appropriate translation lookaside buffer. When the memory management unit is enabled, the effective address is used to locate a TLB entry if found or hit, which will then provide the real address mapping and storage attributes. For performance reasons, a translation lookaside buffer is implemented in each hardware cache to hold recently used address translations. In the MPC823, the table lookup and TLB reload are performed by a software routine with little hardware assistance. This partition simplifies the hardware and gives the system the opportunity to choose the translation table structure.

A TLB hit in multiple entries is avoided during the TLB reload phase. The TLB logic recognizes that the effective page number (EPN) currently loaded into the translation lookaside buffer overlaps another EPN. At least when taking into account the page sizes, subpage validity flags, problem/privileged state, address space ID (ASID), and the SH values of the TLB entries. When such an event occurs, the current EPN is written into the translation lookaside buffer and the entry of the other EPN is invalidated from the translation lookaside buffer.

The memory management unit supports a multiple virtual address space model and, when enabled, each translation is associated with an ASID. In this case, for the translation to be valid, its ASID must be equal to the current address space ID (CASID) that is in effect when an access is performed.

11.2.1 Translation Lookaside Buffer Operation

Two translation lookaside buffers are provided in the MPC823—one for instruction fetches and one for data accesses. The translation lookaside buffer contains pointers to pages in the real memory where data is indexed by the effective page number and it can hold entries with different page sizes. The entry page size controls the number of effective address bits to be compared and the number of least-significant effective address bits that remain untranslated and passes them as least-significant real address bits.

For a 4K page size, four subpage validity flags are supported, thus allowing any combination of 1K subpages to be mapped. For any other page size, all of these flags should have the same value. Programming pages other than 4K pages with different valid bits is considered a programming error. The subpage validity flags can be manipulated to implement effective page sizes of 1K, 2K, 3K, 4K, or any other combination of 1K subpages. However, subpages of an effective page frame must all map to the same real page. During the translation process, the effective address, the processor problem state (MSR_{PR}), and CASID are provided to the translation lookaside buffer. See Figure 11-1 for details. In the translation lookaside buffer, the effective address and CASID are compared with the EPN and ASID of each entry. The CASID is only compared when the matching entry was programmed as nonshared. See **Section 11.6.1.6 MMU Instruction Real Page Number Register** and **Section 11.6.1.7 MMU Data Real Page Number Register** for details.

A successful TLB hit occurs if the incoming effective address matches the EPN stored in a valid TLB entry and the CASID value stored in the M_CASID register matches the entry's ASID field. At the same time, the subpage validity flag is set for the subpage pointed to by the incoming effective address. If a hit is detected, the content of the real page number is concatenated with the appropriate number of least-significant bits from the effective address to form the real address that is then sent to the cache and memory system.

11.3 PROTECTION

Access control is assigned on a page-by-page basis and any further manipulation is conducted on a group basis.

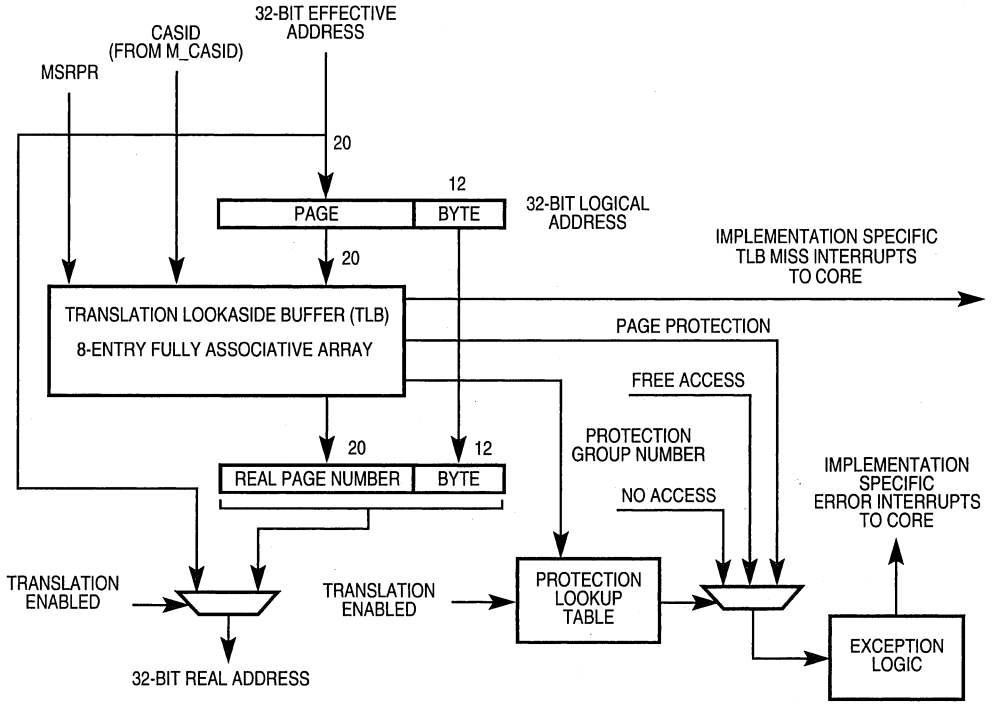


Figure 11-1. Block Diagram of Effective-to-Real Address Translation For 4K Pages

Each TLB entry holds an access protection group (APG) number. When a match is detected, the value of the matched entry's APG is used to index a field in the access protection register that defines access control for the translation. The access protection register contains 16 fields. The field content is used according to the group protection mode. In the PowerPC mode, each field holds the Kp and Ks bits of a corresponding segment register. To be consistent with the *PowerPC Microprocessor Family: The Programming Environment for 32-Bit Microprocessors* manual, the APG value should match the four most-significant bits of the effective page number. In domain manager mode, each field holds override information over the page protection setting. No override, no access override, and free access override modes are all supported.

11.4 STORAGE CONTROL

The memory management unit can be used to map a block of memory in different access modes. Each page can have different storage control attributes. The MPC823 supports cache inhibit, writethrough, and guarded attributes, but not the memory coherence attribute. A page that needs to be memory coherent must be programmed cache-inhibited. Refer to the definition of these attributes in the *PowerPC Microprocessor Family: The Programming Environment for 32-Bit Microprocessors* manual. The effects of the cache-inhibit and writethrough attributes in the MPC823 are described in **Section 9 Instruction Cache**.

The guarded attribute is used to map I/O devices that are sensitive to speculative accesses. An attempt to access a page marked guarded with the guarded bit asserted forces the access to stall until the access is nonspeculative or canceled by the core. Fetching from a guarded storage is prohibited and if it is attempted an implementation-specific instruction storage interrupt is generated. When MSR_{IR} or MSR_{DR} for instruction or data address translation are negated, default attributes are used. See **Section 11.6.1.1 MMU Instruction Control Register** and **Section 11.6.1.2 MMU Data Control Register** for details.

The MPC823 does not generate an exception for a reference bit update because there is no entry for a reference bit in the translation lookaside buffer. The change bit updates are implemented by the software, but the hardware treats the change bit as a write-protect attribute. Therefore, if you try to write to a page marked unmodified, that entry is invalidated and an implementation-specific data TLB error interrupt is generated.

11.5 TRANSLATION TABLE STRUCTURE

The MPC823 memory management unit includes special hardware to assist in a two-level software tablewalk. Other table structures are not precluded. Figure 11-2 and Figure 11-3 illustrate the two levels of translation table structures supported by MPC823 special hardware. When MD_CTR_{TWAM} = 1, the tablewalk begins at the level one base address in the M_TWB register. The level one table is indexed by the ten most-significant bits of the effective address to get the level one page descriptor. For 8M pages, there must be two identical entries in the level one table for either Bit 9 =0 or Bit 9 =1. See Table 11-2 for more information. The level two base address from the level one descriptor is indexed by the next ten least-significant bits to find the level two page descriptor. For pages larger than 4K, the entry in the level two table must be duplicated according to the page size. See Table 11-3 for more information.

During address translation by the memory management unit, the most-significant bits of the missed effective address are replaced by the real page address bits from the level two page descriptor. The number of replaced bits depends on the page size. The rest of the real address bits are taken directly from the effective address. When MD_CTR_{TWAM} = 0, the tablewalk begins at the level one base address placed in the M_TWB register. The level one table is indexed by the 12 most-significant bits of the effective address to get the level one page descriptor. For 8M pages, there must be eight identical entries in the level one table for bits 9-11 of the effective address. The level two base address from the level one descriptor is indexed by the next ten least-significant bits to find the level two page descriptor. For pages larger than 1K, the entry in the level two table must be duplicated according to the page size.

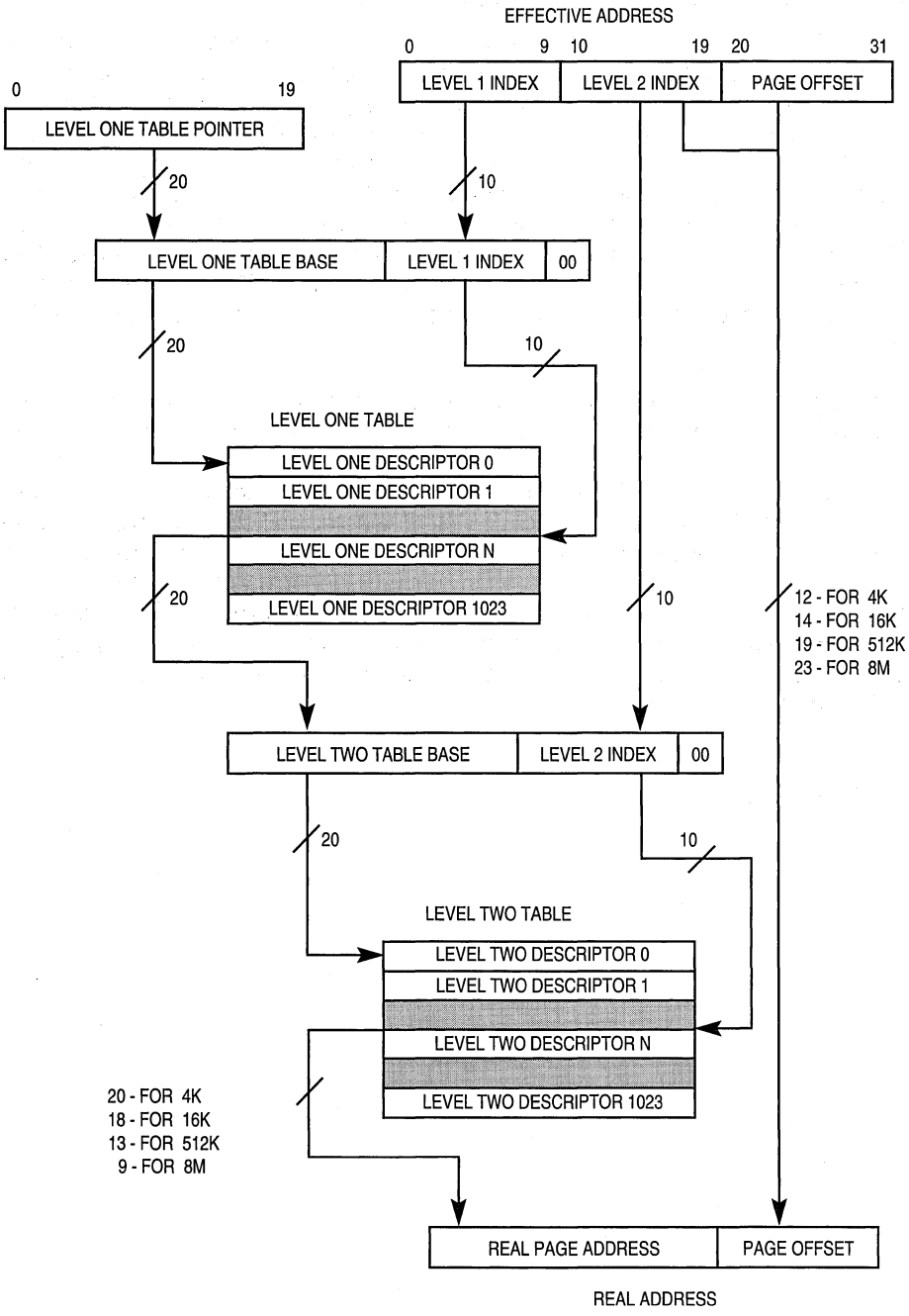


Figure 11-2. Two Level Translation Table When MD_CTR(TWAM) = 1

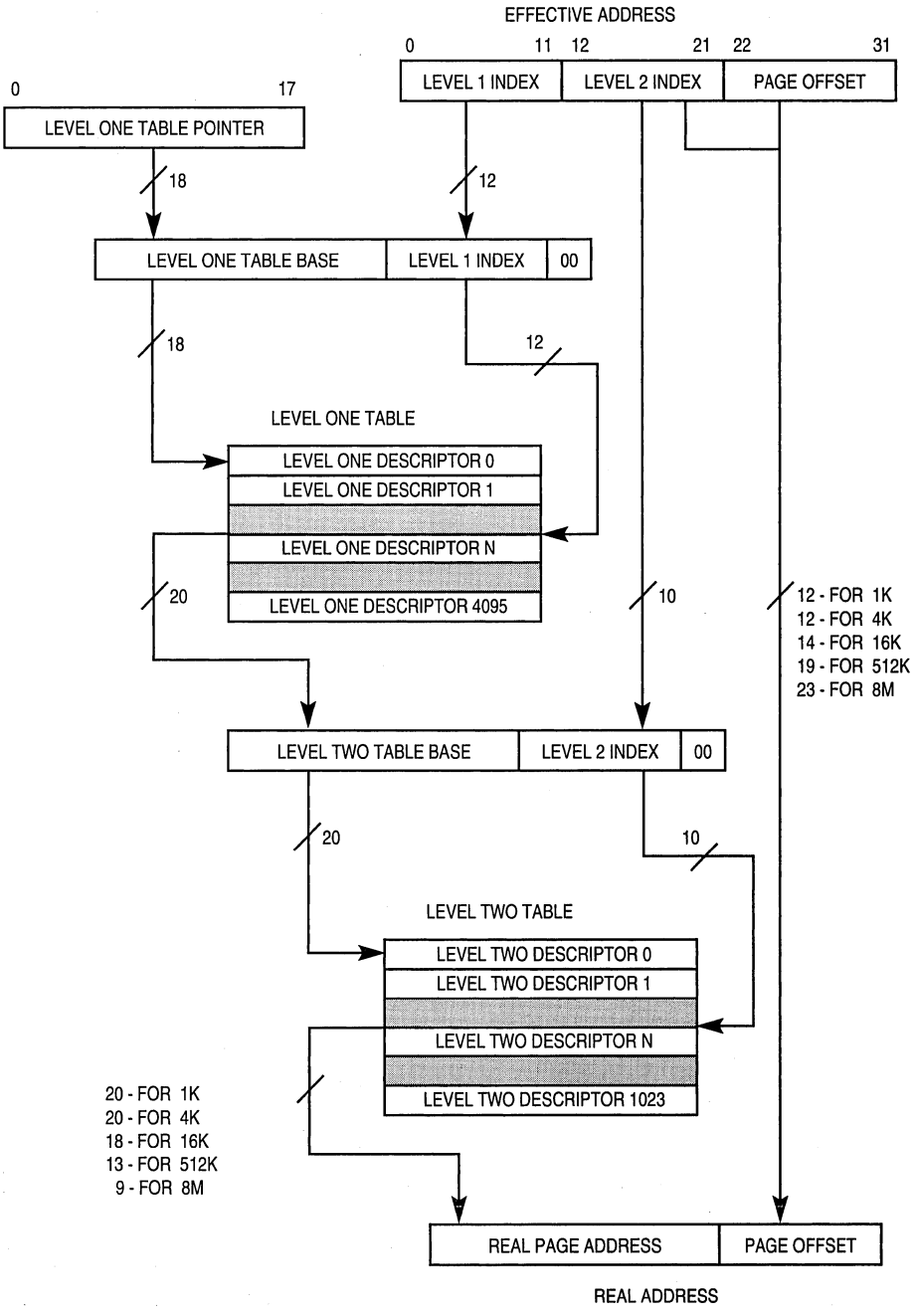


Figure 11-3. Two Level Translation Table When MD_CTR(TWAM) = 0

During the memory management unit's address translation, the most-significant bits of the missed effective address are replaced by the real page address bits from the level two page descriptor. The number of replaced bits depends on the page size. The rest of the real address bits are taken directly from the effective address. See Table 11-1 for details.

Table 11-1. Number of Effective Address Bits Replaced By Real Address Bits

PAGE SIZE	NUMBER OF REPLACED EFFECTIVE ADDRESS BITS
1K	20
4K	20
16K	18
512K	13
8M	9

Table 11-2. Number of Identical Entries Required in the Level One Table

PAGE SIZE	$MD\_CTR_{TWAM} = 0$	$MD\_CTR_{TWAM} = 1$
1K	1	—
4K	1	1
16K	1	1
512K	1	1
8M	8	2

Table 11-3. Number of Identical Entries Required in the Level Two Table

PAGE SIZE	$MD\_CTR_{TWAM} = 0$	$MD\_CTR_{TWAM} = 1$
1K	1	—
4K	4	1
16K	16	4
512K	512	128
8M	1,024	1,024

11.5.1 Level One Descriptor

The following table describes the hardware-assisted level one descriptor format that minimizes the software tablewalk routine.

LEVEL ONE DESCRIPTOR FORMAT

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	L2BA															
RESET	—															
R/W	R/W															
ADDR	SYSTEM MEMORY XXXXX0000															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	L2BA				RESERVED			APG				G	PS		WT	V
RESET	—				—			—				—	—		—	—
R/W	R/W				R/W			R/W				R/W	R/W		R/W	R/W
ADDR	SYSTEM MEMORY XXXXX002															

NOTE: — = Undefined. The default values depend on the state of system memory.

L2BA—Level 2 Table Base Address

This field contains a pointer to a base address of the level 2 table. Bits 18 and 19 are only used when MD_CTR_{TWAM} = 1. Otherwise, they should be set to 0.

Bits 20–22—Reserved

These bits are reserved and should be set to 0.

APG—Access Protection Group

This field contains access protection for the entire memory segment associated with this entry of the table.

G—Guarded Storage Attribute for Entry

0 = Unguarded storage.
1 = Guarded storage.

PS—Page Size Level One

00 = Small (4K or 16K).
01 = 512K.
11 = 8K.
10 = Reserved.

WT—Writethrough Attribute for Entry

0 = Copyback cache policy region (default).
1 = Writethrough cache policy region.

V—Valid

0 = Segment is not valid.
1 = Segment is valid.

11.5.2 Level Two Descriptor

The following table describes the hardware-assisted level two descriptor format that minimizes the software tablewalk routine.

LEVEL TWO DESCRIPTOR FORMAT

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RPN															
RESET	—															
R/W	R/W															
ADDR	SYSTEM MEMORY															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RPN				PP1		PP2		PP3		PP4		SPS	SH	CI	V
RESET	—				—		—		—		—		—	—	—	—
R/W	R/W				R/W		R/W		R/W		R/W		R/W	R/W	R/W	R/W
ADDR	SYSTEM MEMORY															

NOTE: — = Undefined. The default values depend on the state of system memory.

RPN—Real Page Number

This field contains the physical address of the page mapped in by this entry.

PP1—Protection for the First 1K Subpage in a 4K Page

This field contains protection code for the first subpage in a 4K page. Depending on Bit 22 of the PP2 field, the same protection code has different protection schemes.

4K PAGES WITH 1K RESOLUTION PROTECTION				
PP1 SETTING	INSTRUCTION PAGES		DATA PAGES	
	PRIVILEGED MODE	PROBLEM MODE	PRIVILEGED MODE	PROBLEM MODE
00	No access	No access	No access	No access
01	Executable	No access	Read/Write	No access
10	Executable	Executable	Read/Write	Read-only
11	Executable	Executable	Read/Write	Read/Write

PAGES OVER 4K WITH 4K RESOLUTION PROTECTION (EXTENDED ENCODING)				
PP1 SETTING	INSTRUCTION PAGES		DATA PAGES	
	PRIVILEGED MODE	PROBLEM MODE	PRIVILEGED MODE	PROBLEM MODE
00	No access	No access	No access	No access
01	Executable	No access	Read-only	No access
10	Reserved	Reserved	Reserved	Reserved
11	Reserved	Reserved	Reserved	Reserved

PAGES OVER 4K WITH 4K RESOLUTION PROTECTION (POWERPC ENCODING)				
PP1 SETTING	INSTRUCTION PAGES		DATA PAGES	
	PRIVILEGED MODE	PROBLEM MODE	PRIVILEGED MODE	PROBLEM MODE
00	Executable	Executable	Read-only	Read-only
01	Executable	No access	Read-only	No access
10	Executable	Executable	Read/Write	Read-only
11	Executable	Executable	Read/Write	Read/Write

PP2—Protection for a Second 1K Subpage in a 4K Page

This field contains a protection code for the second subpage in a 4K page. Depending on the encoding mode, this field has different meanings.

4K PAGES WITH 1K RESOLUTION PROTECTION				
PP2 SETTING	INSTRUCTION PAGES		DATA PAGES	
	PRIVILEGED MODE	PROBLEM MODE	PRIVILEGED MODE	PROBLEM MODE
00	No access	No access	No access	No access
01	Executable	No access	Read/Write	No access
10	Executable	Executable	Read/Write	Read-only
11	Executable	Executable	Read/Write	Read/Write

PP2 SETTING	PAGES OVER 4K WITH 4K RESOLUTION PROTECTION
0x	PP1 contains PowerPC encoding
1x	PP1 contains Extended Encoding
x1	Page is writable
x0	Page is write-protected

PP3—Protection for the Third 1K Subpage in a 4K Page

This field contains protection code for the third subpage in a 4K page. Depending on the encoding mode, this field has different meanings.

4K PAGES WITH 1K RESOLUTION PROTECTION				
PP3 SETTING	INSTRUCTION PAGES		DATA PAGES	
	PRIVILEGED MODE	PROBLEM MODE	PRIVILEGED MODE	PROBLEM MODE
00	No access	No access	No access	No access
01	Executable	No access	Read/Write	No access
10	Executable	Executable	Read/Write	Read-only
11	Executable	Executable	Read/Write	Read/Write

PAGES OVER 4K WITH 4K RESOLUTION PROTECTION	
PP3 SETTING	CASE: Mx_CTR (PPCS) = 0
0x	First subpage not valid
1x	First subpage valid
x0	Second subpage not valid
x1	Second subpage valid

PAGES OVER 4K WITH 4K RESOLUTION PROTECTION	
PP3 SETTING	**CASE: Mx_CTR (PPCS) = 1**
00	No hit for any state
01	Hit only for problem accesses
10	Hit only for privilege accesses
11	Hit for both problem and privilege accesses

PP4—Protection for the Fourth 1K Subpage in a 4K Page

This field contains protection code for the fourth subpage in a 4K page. Depending on the encoding mode, this field has different meanings.

4K PAGES WITH 1K RESOLUTION PROTECTION				
PP4 SETTING	**INSTRUCTION PAGES**		**DATA PAGES**	
	PRIVILEGED MODE	**PROBLEM MODE**	**PRIVILEGED MODE**	**PROBLEM MODE**
00	No access	No access	No access	No access
01	Executable	No access	Read/Write	No access
10	Executable	Executable	Read/Write	Read-only
11	Executable	Executable	Read/Write	Read/Write

PAGES OVER 4K WITH 4K RESOLUTION PROTECTION	
PP4 SETTING	**CASE: Mx_CTR (PPCS) = 0**
0x	Third subpage not valid
1x	Third subpage valid
x0	Fourth subpage not valid
x1	Fourth subpage valid

PAGES OVER 4K WITH 4K RESOLUTION PROTECTION	
PP4 SETTING	**CASE: Mx_CTR (PPCS) = 1**
00	Must be zero
01	Reserved
10	Reserved
11	Reserved

LPS—Large Page Size

This bit must be set to 0 for 1K resolution protection.

0 = 1K or 4K.
1 = 16K.

SH—Shared Page

0 = This entry matches only if the ASID field in the TLB entry matches the value of the M_CASID register.
1 = ASID comparison is disabled for the entry.

CI—Cache Inhibit

This bit is the cache-inhibit attribute for the entry. Setting this bit will inhibit cache fill for accesses to this page.

V—Valid

This is the page valid bit. Setting this bit indicates the page is valid or resident in the memory (for demand page memory management).

11.6 PROGRAMMING THE MEMORY MANAGEMENT UNIT

The memory management unit implements specific operations using control and status registers, which can be accessed with the PowerPC **mtspr/mfspr** instructions. In addition, the PowerPC **tlbie** and **tlbia** architecture instructions are supported. The memory management unit registers should be accessed when both MSR_{IR} =0 and MSR_{DR} =0. No similar restriction exists for the **tlbie** and **tlbia** instructions.

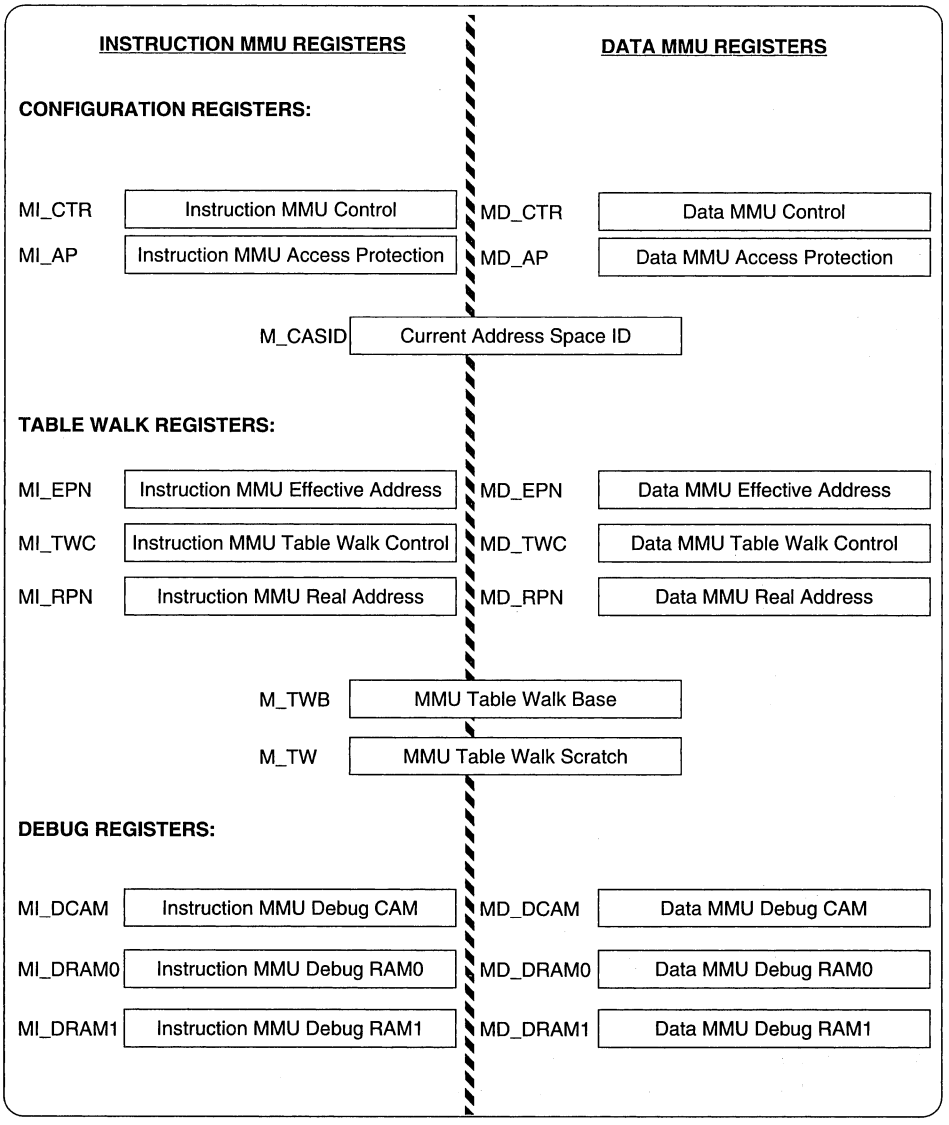


Figure 11-4. Organization of the Memory Management Unit Registers

11.6.1 Control Registers

11.6.1.1 MMU INSTRUCTION CONTROL REGISTER. The MMU instruction control register (MI_CTR) is a special register that is used to control the operation of the instruction memory management unit.

MI_CTR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	GPM	PPM	CIDEF	RES	RSV2I	RES	PPCS	RESERVED								
RESET	0	0	0	0	0	0	0	0								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W								
ADDR	(IMMR & 0xFFFF0000) + 0x784															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED					ITLB_INDX			RESERVED							
RESET	0					0			0							
R/W	R/W					R/W			R/W							
ADDR	(IMMR & 0xFFFF0000) + 0x784															

GPM—Group Protection Mode

0 = PowerPC mode.
1 = Domain manager mode.

PPM—Page Protection Mode

0 = Page resolution protection.
1 = 1K resolution protection for a 4K page.

CIDEF—CI Default

Default value for instruction cache-inhibit attribute when the instruction MMU is disabled (MSR_{IR} = 0).

Bits 3 and 5—Reserved

These bits are reserved and should be set to 0. Ignored on write and returns a 0 on read.

RSV2I—Reserve Two Instruction TLB Entries

0 = ITLB_INDX decremented modulo 8.
1 = ITLB_INDX decremented modulo 6.

PPCS—Privilege/Problem State Compare Mode

0 = Ignore problem/privilege state during address compare.
1 = Consider problem/privilege state according to MI_RPN[24:27].

Bits 7–20—Reserved

These bits are reserved and should be set to 0. Ignored on write and returns a 0 on read.

ITLB_INDX—Instruction TLB Index

This field acts as a pointer to the instruction TLB entry to be loaded. It is automatically decremented at every instruction translation lookaside buffer update.

Bits 24–31—Reserved

These bits are reserved and should be set to 0. Ignored on write and returns a 0 on read.

11.6.1.2 MMU DATA CONTROL REGISTER. The MMU data control register (MD_CTR) is a special register that is used to control the operation of the data memory management unit.

MD_CTR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	GPM	PPM	CIDEF	WTDEF	RSV2D	TWAM	PPCS	RESERVED								
RESET	0	0	0	0	0	0	0	0								
R/W	R/W	R/WR	R/W	R/W	R/W	R/W	R/W	R/W								
ADDR	(IMMR & 0xFFFF0000) + 0x792															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED					DTLB_INDX			RESERVED							
RESET	0					0			0							
R/W	R/W					R/W			R/W							
ADDR	(IMMR & 0xFFFF0000) + 0x792															

GPM—Group Protection Mode

0 = PowerPC mode.
1 = Domain manager mode.

PPM—Page Protection Mode

0 = Page resolution protection.
1 = 1K resolution protection for a 4K page.

CIDEF—CI Default

This bit is the data cache attributes default value when the data MMU is disabled (MSR_{DR} = 0).

WTDEF—WT Default

This bit is the data cache attributes default value when the data MMU is disabled (MSR_{DR} = 0).

RSV2D—Reserve Two Data TLB Entries

0 = DTLB_INDX decremented modulo 8.
1 = DTLB_INDX decremented modulo 6.

TWAM—Tablewalk Assist Mode

0 = 1K subpage hardware assist.
1 = 4K page hardware assist.

PPCS—Privilege/Problem State Compare Mode

0 = Ignore problem/privilege state during address compare.
1 = Consider problem/privilege state according to MD_RPN[24:27].

Bits 7–20—Reserved

These bits are reserved and should be set to 0. Ignored on write and returns a 0 on read.

ITLB_INDX—Instruction TLB Index

This field acts as a pointer to the data TLB entry to be loaded. It is automatically decremented at every data translation lookaside buffer update.

Bits 24–31—Reserved

These bits are reserved and should be set to 0. Ignored on write and returns a 0 on read.

11.6.1.3 MMU CURRENT ADDRESS SPACE ID REGISTER. The MMU current address space ID (M_CASID) register is used to compare the current effective address with the ASID field in the TLB entry when searching for a matching entry.

M_CASID

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
RESET	—															
R/W	R															
ADDR	(IMMR & 0xFFFF0000) + 0x793															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED												CASID			
RESET	—												—			
R/W	R												R/W			
ADDR	(IMMR & 0xFFFF0000) + 0x793															

NOTE: — = Undefined.

Bits 0–27—Reserved

These bits are reserved and should be set to 0. Ignored on a write.

CASID—Current Address Space ID

This field is compared to the ASID field of a TLB entry to qualify a match.

11.6.1.4 MMU INSTRUCTION EFFECTIVE PAGE NUMBER REGISTER. The MMU instruction effective page number (MI_EPN) register contains the effective address to be loaded into a TLB entry.

MI_EPN

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	EPN															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x787															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	EPN				RESERVED		EV	RESERVED					ASID			
RESET	—				—		0	0					0			
R/W	R/W				R		R/W	R					R/W			
ADDR	(IMMR & 0xFFFF0000) + 0x787															

NOTE: — = Undefined.

EPN—Effective Page Number for the TLB Entry

This field is the effective address default value of the last instruction TLB miss.

Bits 20–21 and 23–27—Reserved

These bits are reserved and should be set to 0. Ignores on write and returns a 0 on read.

EV—TLB Entry Valid Bit

This bit is set to 1 on every instruction TLB miss.

0 = The TLB entry is invalid.
1 = The TLB entry is valid.

ASID—Address Space ID

This field represent the address space ID of the instruction TLB entry to be compared with the CASID field of the M_CASID register.

11.6.1.5 MMU DATA EFFECTIVE PAGE NUMBER REGISTER. The MMU data effective page number (MD_EPN) register contains the effective address to be loaded into a TLB entry.

MD_EPN

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	EPN															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x795															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	EPN						EV	RESERVED					ASID			
RESET	—						—	0					—			
R/W	R/W						R/W	R/W					R/W			
ADDR	(IMMR & 0xFFFF0000) + 0x795															

NOTE: — = Undefined.

EPN—Effective Page Number for Entry

The default value is the effective address of the last data TLB miss.

EV—TLB Entry Valid

This bit is set to 1 on a data TLB miss.

0 = The data TLB entry is invalid.
1 = The data TLB entry is valid.

Bits 23–27—Reserved

These bits are reserved and should be set to 0. Ignores on write and returns a 0 on read.

ASID—Address Space ID

This field is the address space IDs of the TLB entry to be compared with the CASID field of the M_CASID register.

11.6.1.6 MMU INSTRUCTION REAL PAGE NUMBER REGISTER. The MMU instruction real page number (MI_RPN) register contains the physical address and the storage attributes of an entry to be loaded into a translation lookaside buffer. This register should be written after the MI_EPN and MI_TWC registers are written.

MI_RPN

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RPN															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x790															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RPN				PP1		PP2		PP3		PP4		LPS	SH	CI	V
RESET	—				—		—		—		—		—	—	—	—
R/W	R/W				R/W		R/W		R/W		R/W		R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x790															

NOTE: — = Undefined. The default values depend on the state of system memory.

RPN—Real Page Number

These bits are the most-significant bits of the page's physical address.

PP1—Protection for the First 1K Subpage in a 4K Page

This field contains protection code for the first subpage in a 4K page. Depending on Bit 22 of the PP2 field, the same protection code has different protection schemes.

4K PAGES WITH 1K RESOLUTION PROTECTION		
PP1 SETTING	INSTRUCTION PAGES	
	PRIVILEGED MODE	PROBLEM MODE
00	No access	No access
01	Executable	No access
10	Executable	Executable
11	Executable	Executable

PAGES OVER 4K WITH 4K RESOLUTION PROTECTION (EXTENDED ENCODING)		
PP1 SETTING	INSTRUCTION PAGES	
	PRIVILEGED MODE	PROBLEM MODE
00	No access	No access
01	Executable	No access
10	Reserved	Reserved
11	Reserved	Reserved

PAGES OVER 4K WITH 4K RESOLUTION PROTECTION (POWERPC ENCODING)		
PP1 SETTING	INSTRUCTION PAGES	
	PRIVILEGED MODE	PROBLEM MODE
00	Executable	Executable
01	Executable	No access
10	Executable	Executable
11	Executable	Executable

PP2—Protection for a Second 1K Subpage in a 4K Page

This field contains a protection code for the second subpage in a 4K page. Depending on the encoding mode, this field has different meanings.

4K PAGES WITH 1K RESOLUTION PROTECTION		
PP2 SETTING	INSTRUCTION PAGES	
	PRIVILEGED MODE	PROBLEM MODE
00	No access	No access
01	Executable	No access
10	Executable	Executable
11	Executable	Executable

PP2 SETTING	PAGES OVER 4K WITH 4K RESOLUTION PROTECTION
0x	PP1 contains PowerPC encoding
1x	PP1 contains Extended Encoding
x1	Page is writable
x0	Page is write-protected

PP3—Protection for the Third 1K Subpage in a 4K Page

This field contains protection code for the third subpage in a 4K page. Depending on the encoding mode, this field has different meanings.

4K PAGES WITH 1K RESOLUTION PROTECTION		
PP3 SETTING	INSTRUCTION PAGES	
	PRIVILEGED MODE	PROBLEM MODE
00	No access	No access
01	Executable	No access
10	Executable	Executable
11	Executable	Executable

PAGES OVER 4K WITH 4K RESOLUTION PROTECTION	
PP3 SETTING	CASE: MI_CTR (PPCS) = 0
0x	First subpage not valid
1x	First subpage valid
x0	Second subpage not valid
x1	Second subpage valid

PAGES OVER 4K WITH 4K RESOLUTION PROTECTION	
PP3 SETTING	**CASE: MI_CTR (PPCS) = 1**
00	No hit for any state
01	Hit only for problem accesses
10	Hit only for privilege accesses
11	Hit for both problem and privilege accesses

PP4—Protection for the Fourth 1K Subpage in a 4K Page

This field contains protection code for the fourth subpage in a 4K page. Depending on the encoding mode, this field has different meanings.

4K PAGES WITH 1K RESOLUTION PROTECTION		
PP4 SETTING	**INSTRUCTION PAGES**	
	PRIVILEGED MODE	**PROBLEM MODE**
00	No access	No access
01	Executable	No access
10	Executable	Executable
11	Executable	Executable

PAGES OVER 4K WITH 4K RESOLUTION PROTECTION	
PP4 SETTING	**CASE: MI_CTR (PPCS) = 0**
0x	Third subpage not valid
1x	Third subpage valid
x0	Fourth subpage not valid
x1	Fourth subpage valid

PAGES OVER 4K WITH 4K RESOLUTION PROTECTION	
PP4 SETTING	**CASE: MI_CTR (PPCS) = 1**
00	Must be zero
01	Reserved
10	Reserved
11	Reserved

LPS—Large Page Size

This bit must be set to 0 for 1K resolution protection.

0 = 1K or 4K.
1 = 16K.

SH—Shared Page

0 = This entry matches only if the ASID filed in the TLB entry matches the value of the M_CASID register.
1 = ASID comparison is disabled for the entry.

CI—Cache Inhibit

This bit is the cache-inhibit attribute for the TLB entry.

V—Valid

This bit indicates that a TLB entry is valid.

11.6.1.7 MMU DATA REAL PAGE NUMBER REGISTER. The MMU data real page number (MD_RPN) register contains the physical address and the storage attributes of an entry to be loaded into a translation lookaside buffer. This register should be written after the MD_EPN and MD_TWC registers.

MD_RPN

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RPN															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x798															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RPN				PP1		PP2		PP3		PP4		LPS	SH	CI	V
RESET	—				—		—		—		—		—	—	—	—
R/W	R/W				R/W		R/W		R/W		R/W		R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x798															

NOTE: — = Undefined. The default values depend on the state of system memory.

RPN—Real Page Number

These bits are the most-significant bits of the page's physical address.

PP1—Protection for the First 1K Subpage in a 4K Page

This field contains protection code for the first subpage in a 4K page. Depending on Bit 22 of the PP2 field, the same protection code has different protection schemes.

4K PAGES WITH 1K RESOLUTION PROTECTION		
PP1 SETTING	DATA PAGES	
	PRIVILEGED MODE	PROBLEM MODE
00	No access	No access
01	Read/Write	No access
10	Read/Write	Read-only
11	Read/Write	Read/Write

PAGES OVER 4K WITH 4K RESOLUTION PROTECTION (EXTENDED ENCODING)		
PP1 SETTING	DATA PAGES	
	PRIVILEGED MODE	PROBLEM MODE
00	No access	No access
01	Read-only	No access
10	Reserved	Reserved
11	Reserved	Reserved

PAGES OVER 4K WITH 4K RESOLUTION PROTECTION (POWERPC ENCODING)		
PP1 SETTING	DATA PAGES	
	PRIVILEGED MODE	PROBLEM MODE
00	Read-only	Read-only
01	Read-only	No access
10	Read/Write	Read-only
11	Read/Write	Read/Write

PP2—Protection for a Second 1K Subpage in a 4K Page

This field contains a protection code for the second subpage in a 4K page. Depending on the encoding mode, this field has different meanings.

4K PAGES WITH 1K RESOLUTION PROTECTION		
PP2 SETTING	DATA PAGES	
	PRIVILEGED MODE	PROBLEM MODE
00	No access	No access
01	Read/Write	No access
10	Read/Write	Read-only
11	Read/Write	Read/Write

PP2 SETTING	PAGES OVER 4K WITH 4K RESOLUTION PROTECTION
0x	PP1 contains PowerPC encoding
1x	PP1 contains Extended Encoding
x1	Page is writable
x0	Page is write-protected

PP3—Protection for the Third 1K Subpage in a 4K Page

This field contains protection code for the third subpage in a 4K page. Depending on the encoding mode, this field has different meanings.

4K PAGES WITH 1K RESOLUTION PROTECTION		
PP3 SETTING	DATA PAGES	
	PRIVILEGED MODE	PROBLEM MODE
00	No access	No access
01	Read/Write	No access
10	Read/Write	Read-only
11	Read/Write	Read/Write

PAGES OVER 4K WITH 4K RESOLUTION PROTECTION	
PP3 SETTING	CASE: MD_CTR (PPCS) = 0
0x	First subpage not valid
1x	First subpage valid
x0	Second subpage not valid
x1	Second subpage valid

PAGES OVER 4K WITH 4K RESOLUTION PROTECTION	
PP3 SETTING	**CASE: MD_CTR (PPCS) = 1**
00	No hit for any state
01	Hit only for problem accesses
10	Hit only for privilege accesses
11	Hit for both problem and privilege accesses

PP4—Protection for the Fourth 1K Subpage in a 4K Page

This field contains protection code for the fourth subpage in a 4K page. Depending on the encoding mode, this field has different meanings.

4K PAGES WITH 1K RESOLUTION PROTECTION		
PP4 SETTING	**DATA PAGES**	
	PRIVILEGED MODE	**PROBLEM MODE**
00	No access	No access
01	Read/Write	No access
10	Read/Write	Read-only
11	Read/Write	Read/Write

PAGES OVER 4K WITH 4K RESOLUTION PROTECTION	
PP4 SETTING	**CASE: MD_CTR (PPCS) = 0**
0x	Third subpage not valid
1x	Third subpage valid
x0	Fourth subpage not valid
x1	Fourth subpage valid

PAGES OVER 4K WITH 4K RESOLUTION PROTECTION	
PP4 SETTING	**CASE: MD_CTR (PPCS) = 1**
00	Must be zero
01	Reserved
10	Reserved
11	Reserved

LPS—Large Page Size

This bit must be set to 0 for 1K resolution protection.

0 = 1K or 4K.
1 = 16K.

SH—Shared Page

0 = This entry matches only if the ASID filed in the TLB entry matches the value of the M_CASID register.
1 = ASID comparison is disabled for a TLB entry.

CI—Cache Inhibit

This bit is the cache-inhibit attribute for a TLB entry.

V—Valid

This bit indicates that a TLB entry is valid.

11.6.1.8 MMU INSTRUCTION ACCESS PROTECTION REGISTER. The MMU instruction access protection (MI_AP) register contains the access protection group for the instruction memory management unit.

MI_AP

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	GP0		GP1		GP2		GP3		GP4		GP5		GP6		GP7	
RESET	—		—		—		—		—		—		—		—	
R/W	R/W		R/W		R/W		R/W		R/W		R/W		R/W		R/W	
ADDR	(IMMR & 0xFFFF0000) + 0x786															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	GP8		GP9		GP10		GP11		GP12		GP13		GP14		GP15	
RESET	—		—		—		—		—		—		—		—	
R/W	R/W		R/W		R/W		R/W		R/W		R/W		R/W		R/W	
ADDR	(IMMR & 0xFFFF0000) + 0x786															

NOTE: — = Undefined.

GPx—Group Protection

In domain manager mode, these bits have the following settings.

00 = No access.
01 = Client-access permission defined by page protection bits.
10 = Reserved.
11 = Manager-free access.

In PowerPC mode, the GP bits have these settings and are privilege and problem state (Ks and Kp) in the *PowerPC Microprocessor Family: The Programming Environment for 32-Bit Microprocessors* manual:

00 = All accesses are considered privileged.
01 = Access permission defined by page protection bits.
10 = Problem and privileged interpretation is swapped.
11 = All accesses are considered problem.

11.6.1.9 MMU DATA ACCESS PROTECTION REGISTER. The MMU data access protection (MD_AP) register contains the access protection group for the data memory management unit.

MD_AP

BIT	0 1	2 3	4 5	6 7	8 9	10 11	12 13	14 15
FIELD	GP0	GP1	GP2	GP3	GP4	GP5	GP6	GP7
RESET	—	—	—	—	—	—	—	—
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x794							
BIT	16 17	18 19	20 21	22 23	24 25	26 27	28 29	30 31
FIELD	GP8	GP9	GP10	GP11	GP12	GP13	GP14	GP15
RESET	—	—	—	—	—	—	—	—
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x794							

NOTE: — = Undefined.

GPx—Group Protection

In domain manager mode, these bits have the following settings.

00 = No access.
01 = Client-access permission defined by page protection bits.
10 = Reserved.
11 = Manager-free access.

In PowerPC mode, the GP bits have these settings and are privilege and problem state (Ks and Kp) in the *PowerPC Microprocessor Family: The Programming Environment for 32-Bit Microprocessors* manual:

00 = All accesses are considered privileged.
01 = Access permission defined by page protection bits.
10 = Problem and privileged interpretation is swapped.
11 = All accesses are considered problem.

11.6.1.10 MMU INSTRUCTION TABLEWALK CONTROL REGISTER. The MMU instruction tablewalk control (MI_TWC) register contains the access protection group and page size of the entry to be loaded into the translation lookaside buffer.

MI_TWC

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x789															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED							APG				G	PS		RES	V
RESET	0							—				—	—		0	—
R/W	R/W							R/W				R/W	R/W		R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x789															

NOTE: — = Undefined.

Bits 0–22 and 30—Reserved

These bits are reserved and should be set to 0. Ignores on write and returns a 0 on read.

APG—Access Protection Group

A maximum of 16 protection groups are supported. The default value of instruction TLB miss is 0.

G—Guarded Storage Attribute for Entry

Default value on instruction TLB miss is 0.

0 = Unguarded storage.
1 = Guarded storage.

PS—Page Size Level One

Default value on instruction TLB miss is 00.

00 = Small (4K or 16K).
01 = 512K.
11 = 8M.
10 = Reserved.

V—Entry Valid

Default value on instruction TLB miss is 1.

0 = Entry is not valid.
1 = Entry is valid.

11.6.1.11 MMU DATA TABLEWALK CONTROL REGISTER. The MMU data tablewalk control (MD_TWC) register contains the second level pointer and access protection group of an entry to be loaded into the translation lookaside buffer.

MD_TWC

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	L2TB															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x797															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	L2TB				RESERVED			APG				G	PS		WT	V
RESET	—				—			—				—	—		—	—
R/W	R/W				R/W			R/W				R/W	R/W		R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x797															

NOTE: — = Undefined.

L2TB—Tablewalk Level 2 Base Value

These bits are the most-significant bits of the level two pointer.

Bits 20–22—Reserved

When written, these bits are reserved and should be set to 0. When read, they return MD_EPN[10:19] when MD_CTR_{TWAM} = 1 and MD_EPN[12:21] when MD_CTR_{TWAM} = 0.

APG—Access Protection Group

When written, this field supports a maximum of 16 protection groups. It is set to 0000 on the data TLB miss. When read, it returns MD_EPN[10:19] when MD_CTR_{TWAM} = 1 and MD_EPN[12:21] when MD_CTR_{TWAM} = 0.

G—Guarded

When written, this bit of the entry has the following settings and is set to 0 on a data TLB miss:

0 = Unguarded storage.
1 = Guarded storage.

When read, these bit returns MD_EPN[10:19] when MD_CTR_{TWAM} = 1 and MD_EPN[12:21] when MD_CTR_{TWAM} = 0.

PS—Page Size

When written, this field has the following settings and is set to 0 on a data TLB miss. When this field is read, it returns a zero.

00 = Small (4K or 16K).
01 = 512K.
11 = 8M.
10 = Reserved.

When read, this field returns MD_EPN[10:19] when MD_CTR_{TWAM} = 1 and MD_EPN[12:21] when MD_CTR_{TWAM} = 0.

WT—Writethrough

When written, this bit has the following settings and is set to 1 on a data TLB miss. When this bit is read, it returns a zero.

0 = Copyback data cache policy page entry.
1 = Writethrough data cache policy page entry.

V—Valid

When written, this bit has the following settings and is set to 1 on a data TLB miss. When this bit is read, it returns a zero.

0 = Entry is invalid.
1 = Entry is valid.

11.6.1.12 MMU TABLEWALK BASE REGISTER. The MMU tablewalk base (M_TWB) register contains a pointer to the level one table to be used in hardware-assisted tablewalk mode.

M_TWB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	L1TB															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x796															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	L1TB				L1INDX										RESERVED	
RESET	—				—										0	
R/W	R/W				R/W										R/W	
ADDR	(IMMR & 0xFFFF0000) + 0x796															

NOTE: — = Undefined.

L1TB—Tablewalk Level 1 Base Value

These bits are the most-significant bits of the level one pointer.

L1INDX—Level 1 Table Index

This field is ignored on write. It returns MD_EPN[0:9] on read when MD_CTR_{TWAM} = 1 and MD_EPN[2:11] when MD_CTR_{TWAM} = 0.

Bits 30–31—Reserved

These bits are reserved and should be set to 0. Ignores on write and returns a 0 on read.

11.6.1.13 MMU TABLEWALK SPECIAL REGISTER. The MMU tablewalk special (M_TW) register is temporarily used in the software tablewalk interrupt handlers.

M_TW

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x799															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x799															

NOTE: — = Undefined.

11.6.2 MMU Data Content-Addressable Registers

The MD_CAM, MD_RAM0, and MD_RAM1 registers are interface registers that allow you to read data memory management unit CAM and RAM entries. If you try to write to the MD_CAM register using the **mtspr** instruction, the CAM and RAM values of the entry indexed by the DTLB_INDX field to MD_CAM, MD_RAM0, and MD_RAM1 will be loaded. The source register in the **mtspr** instruction can be any register, since its value is not used. The values of the MD_CAM, MD_RAM0, and MD_RAM1 registers can be read using the **mtspr** instruction. If you try to write to the MD_RAM0 and MD_RAM1 registers using the **mtspr** instruction, it will be considered a NOP (no operation) instruction.

11.6.2.1 MMU DATA CAM ENTRY READ REGISTER. When the content-addressable memory of the MMU data CAM entry read (MD_CAM) register is read, it contains the effective address and page sizes of an entry indexed by the DTLB_INDX field of the MD_CAM. This register is only updated when you write a value to it.

MD_CAM

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	EPN															
RESET	—															
R/W	R															
ADDR	(IMMR & 0xFFFF0000) + 0x824															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	EPN				SPVF				PS			SH	ASID			
RESET	—				—				—			—	—			
R/W	R				R				R			R	R			
ADDR	(IMMR & 0xFFFF0000) + 0x824															

NOTE: — = Undefined.

EPN—Effective Page Number

These bits are the most-significant bits of the page's effective address.

SPVF—Subpage Validity Flags

For Bit 20:
0 = Subpage 0 (address[20:21] = 00) is not valid.
1 = Subpage 0 (address[20:21] = 00) is valid.

For Bit 21:
0 = Subpage 1 (address[20:21] = 01) is not valid.
1 = Subpage 1 (address[20:21] = 01) is valid.

For Bit 22:
0 = Subpage 2 (address[20:21] = 10) is not valid.
1 = Subpage 2 (address[20:21] = 10) is valid.

For Bit 23:
0 = Subpage 3 (address[20:21] = 11) is not valid.
1 = Subpage 3 (address[20:21] = 11) is valid.

PS—Page Size

000 = 4K.
001 = 16K.
011 = 512K.
111 = 8M.
010 = Reserved.
100 = Reserved.
101 = Reserved.
110 = Reserved.

SH—Shared Page

0 = This entry matches only if the ASID field in the data TLB entry matches the value of the M_CASID register.
1 = ASID comparison is disabled for the entry.

ASID—Address Space ID

This field is the address space ID of the TLB entry to be compared with the CASID field of the M_CASID register.

11.6.2.2 MMU DATA RAM ENTRY READ REGISTER 0. The MMU data RAM entry read register 0 (MD_RAM0) contains the physical page number and page attributes of an entry indexed by the DTLB_INDX field of the MD_CTR. This register is only updated when you write a value to it.

MD_RAM0

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RPN															
RESET	—															
R/W	R															
ADDR	(IMMR & 0xFFFF0000) + 0x825															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RPN				PS			APGI				G	WT	CI	RESERVED	
RESET	—				—			—				—	—	—	0	
R/W	R				R			R				R	R	R	R	
ADDR	(IMMR & 0xFFFF0000) + 0x825															

NOTE: — = Undefined.

RPN—Real Page Number

These bits are the most-significant bits of the page's physical address.

PS—Page Size

000 = 4K.
001 = 16K.
011 = 512K.
111 = 8M.
010 = Reserved.
100 = Reserved.
101 = Reserved.
110 = Reserved.

APGI—Access Protection Group Inverted

This field is an inversion of the access protection group in the MD_TWC register.

G—Guarded

0 = Unguarded storage.
1 = Guarded storage.

WT—Writethrough

0 = Copyback data cache policy page entry.
1 = Writethrough data cache policy page entry.

CI—Cache-Inhibit

When this bit is 0, it is not cache inhibited.

Bits 30–31—Reserved

These bits are reserved and should be set to 0.

11.6.2.3 MMU DATA RAM ENTRY READ REGISTER 1. The MMU data RAM entry read register 1 (MD_RAM1) contains the protection mode information of the entry indexed by the DTLB_INDX field of the MD_CTR. This register is only updated when you write a value to it.

MD_RAM1

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
RESET	0															
R/W	R															
ADDR	(IMMR & 0xFFFF0000) + 0x826															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RES	C	EVF	SA				SAT	URP0	UWP0	URP1	UWP1	URP2	UWP2	URP3	UWP3
RESET	0	—	—	—				—	—	—	—	—	—	—	—	—
R/W	R	R	R	R				R	R	R	R	R	R	R	R	R
ADDR	(IMMR & 0xFFFF0000) + 0x826															

NOTE: — = Undefined.

Bits 0–16—Reserved

These bits are reserved and should be set to 0.

C—Change Bit for Data Entry TLB

0 = Unchanged region. Write access to this page results in the implementation-specific instruction MMU interrupt invocation. software should take an appropriate action before setting this bit to 1.

1 = Changed region. Write access is allowed to this page.

EVF—Entry Valid Flag

0 = Entry is invalid.

1 = Entry is valid.

SA—Privileged (Supervisor) Access

For Bit 19:

0 = Subpage 0 (address[20:21]=00) privileged access is not permitted.

1 = Subpage 0 (address[20:21]=00) privileged access is permitted.

For Bit 20:

0 = Subpage 1 (address[20:21]=01) privileged access is not permitted.

1 = Subpage 1 (address[20:21]=01) privileged access is permitted.

For Bit 21:
0 = Subpage 2 (address[20:21]=10) privileged access is not permitted.
1 = Subpage 2 (address[20:21]=10) privileged access is permitted.

For Bit 22:
0 = Subpage 3 (address[20:21]=11) privileged access is not permitted.
1 = Subpage 3 (address[20:21]=11) privileged access is permitted.

SAT—Privileged (Supervisor) Access Type
0 = Privileged access type is read-only.
1 = Privileged access type is read/write.

URP0—Problem (User) Read Permission Page 0
0 = Subpage 0 (address[20:21]=00) problem read access is not permitted.
1 = Subpage 0 (address[20:21]=00) problem read access is permitted.

UWP0—Problem (User) Read Permission Page Zero
0 = Subpage 0 (address[20:21]=00) problem write access is not permitted.
1 = Subpage 0 (address[20:21]=00) problem write access is permitted.

URP1—Problem (User) Read Permission Page 1
0 = Subpage 1 (address[20:21]=01) problem read access is not permitted.
1 = Subpage 1 (address[20:21]=01) problem read access is permitted.

URP2—Problem (User) Read Permission Page Two
0 = Subpage 2 (address[20:21]=10) problem read access is not permitted.
1 = Subpage 2 (address[20:21]=10) problem read access is permitted.

UWP2—Problem (User) Write Permission Page Two
0 = Subpage 2 (address[20:21]=10) problem write access is not permitted.
1 = Subpage 2 (address[20:21]=10) problem write access is permitted.

URP3—Problem (User) Read Permission Page Three
0 = Subpage 3 (address[20:21]=11) problem read access is not permitted.
1 = Subpage 3 (address[20:21]=11) problem read access is permitted.

UWP3—Problem (User) Write Permission Page Three
0 = Subpage 3 (address[20:21]=11) problem write access is not permitted.
1 = Subpage 3 (address[20:21]=11) problem write access is permitted.

11.6.3 MMU Instruction Content-Addressable Registers

The MI_CAM, MI_RAM0, and MI_RAM1 registers are interface registers that allow you to read data memory management unit CAM and RAM entries. If you try to write to the MI_CAM register using the **mtspr** instruction, the CAM and RAM values of the entry indexed by the DTLB_INDX field to MI_CAM, MI_RAM0, and MI_RAM1 will be loaded. The source register in the **mtspr** instruction can be any register, since its value is not used. The values of the MI_CAM, MI_RAM0, and MI_RAM1 registers can be read using the **mtspr** instruction. If you try to write to the MI_RAM0 and MI_RAM1 registers using the **mtspr** instruction, it will be considered a NOP (no operation) instruction.

11.6.3.1 MMU INSTRUCTION CAM ENTRY READ REGISTER. When the content-addressable memory of the MMU instruction CAM entry read (MI_CAM) register is read, it contains the effective address and page sizes of an entry indexed by the ITLB_INDX field of the MI_CTR. This register is only updated when you write a value to it.

MI_CAM

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	EPN															
RESET	—															
R/W	R															
ADDR	(IMMR & 0xFFFF0000) + 0x816															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	EPN				PS			ASID				SH	SPV			
RESET	—				—			—				—	—			
R/W	R				R			R				R	R			
ADDR	(IMMR & 0xFFFF0000) + 0x816															

NOTE: — = Undefined.

EPN—Effective Page Number

These bits are the most-significant bits of the page's effective address.

PS—Page Size

000 = 4K.
001 = 16K.
011 = 512K.
111 = 8M.
010 = Reserved.
100 = Reserved.
101 = Reserved.
110 = Reserved.

ASID—Address Space ID

This field represents the data TLB entry to be compared with the CASID field in the M_CASID register.

SH—Shared Page

0 = This entry matches only if the ASID field in the data TLB entry matches the value of the M_CASID register.
1 = ASID comparison is disabled for the entry.

SPV—Subpage Validity

Bit 28:
0 = Subpage 0 (address[20:21]=00) is not valid.
1 = Subpage 0 (address[20:21]=00) is valid.

Bit 29:
0 = Subpage 1 (address[20:21]=01) is not valid.
1 = Subpage 1 (address[20:21]=01) is valid.

Bit 30:
0 = Subpage 2 (address[20:21]=10) is not valid.
1 = Subpage 2 (address[20:21]=10) is valid.

Bit 31:
0 = Subpage 3 (address[20:21]=11) is not valid.
1 = Subpage 3 (address[20:21]=11) is valid.

11.6.3.2 MMU INSTRUCTION RAM ENTRY READ REGISTER 0. The MMU instruction RAM entry read register 0 (MI_RAM0) contains the physical page number and page attributes of an entry indexed by the ITLB_INDX field of the MI_CTR. This register is only updated when you write to the MI_CAM register.

MI_RAM0

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RPN															
RESET	—															
R/W	R															
ADDR	(IMMR & 0xFFFF0000) + 0x817															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RPN				PS_B			CI	APG				SFP			
RESET	—				—			—	—				—			
R/W	R				R			R	R				R			
ADDR	(IMMR & 0xFFFF0000) + 0x817															

NOTE: — = Undefined.

RPN—Real Page Number

These bits are the most-significant bits of the page's physical address.

PS_B—Page Size

000 = 4K.
001 = 16K.
011 = 512K.
111 = 8M.
010 = Reserved.
100 = Reserved.
101 = Reserved.
110 = Reserved.

CI—Cache-Inhibit

When this bit is 0, it is not cache-inhibited.

APG—Access Protection Group

A maximum of 16 protection groups are supported and are represented in one's compliment format.

SFP—Privileged (Supervisor) Fetch Permission

Bit 28:
0 = Subpage 0 (address[20:21]=00) privileged fetch is not permitted.
1 = Subpage 0 (address[20:21]=00) privileged fetch is permitted.

Bit 29:
0 = Subpage 1 (address[20:21]=01) privileged fetch is not permitted.
1 = Subpage 1 (address[20:21]=01) privileged fetch is permitted.

Bit 30:
0 = Subpage 2 (address[20:21]=10) privileged fetch is not permitted.
1 = Subpage 2 (address[20:21]=10) privileged fetch is permitted.

Bit 31:
0 = Subpage 3 (address[20:21]=11) privileged fetch is not permitted.
1 = Subpage 3 (address[20:21]=11) privileged fetch is permitted.

11.6.3.3 MMU INSTRUCTION RAM ENTRY READ REGISTER 1. The MMU instruction RAM entry read register 1 (MI_RAM1) contains the protection mode information of the entry indexed by the ITLB_INDX field of the MI_CTR. This register is only updated when you write to the MI_CAM register.

MI_RAM1

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
RESET	0															
R/W	R															
ADDR	(IMMR & 0xFFFF0000) + 0x818															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED										UFP				PV	G
RESET	0										—				—	—
R/W	R										R				R	R
ADDR	(IMMR & 0xFFFF0000) + 0x818															

NOTE: — = Undefined.

Bits 0–25—Reserved
These bits are reserved and should be set to zero.

UFP—Problem (User) Fetch Permission

Bit 26:
0 = Subpage 0 (address[20:21]=00) problem fetch is not permitted.
1 = Subpage 0 (address[20:21]=00) problem fetch is permitted.

Bit 27:
0 = Subpage 1 (address[20:21]=01) problem fetch is not permitted.
1 = Subpage 1 (address[20:21]=01) problem fetch is permitted.

Bit 28:
0 = Subpage 2 (address[20:21]=10) problem fetch is not permitted.
1 = Subpage 2 (address[20:21]=10) problem fetch is permitted.

Bit 29:
0 = Subpage 3 (address[20:21]=11) problem fetch is not permitted.
1 = Subpage 3 (address[20:21]=11) problem fetch is permitted.

PV—Page Validity

0 = Page is invalid.
1 = Page is valid.

G—Guarded

0 = Unguarded storage.
1 = Guarded storage.

11.7 INTERRUPTS

11.7.1 Implementation-Specific Instruction TLB Miss

The implementation-specific instruction TLB miss interrupt occurs when MSR_{IR} =1 and there is an attempt to fetch an instruction from a page whose effective page number cannot be found in the instruction translation lookaside buffer. The software tablewalk code is responsible for loading the translation information of the missed page from the translation table that resides in the memory. Refer to **Section 11.8.1.1 Translation Reload Examples** for more information.

11.7.2 Implementation-Specific Data TLB Miss

The implementation-specific data TLB miss interrupt occurs when MSR_{DR} =1 and there is an attempt to access a page whose effective page number cannot be found in the data translation lookaside buffer. The software tablewalk code is responsible for loading the translation information of the missed page from the translation table that resides in the memory. Refer to **Section 11.8.1.1 Translation Reload Examples** for more information.

11.7.3 Implementation-Specific Instruction TLB Error

The implementation-specific instruction TLB error interrupt occurs under one of the following conditions:

- The effective address cannot be translated. Either the segment or page valid bit of this page is cleared in the translation table.
- The fetch access violates storage protection.
- The fetch access is to guarded storage and MSR_{IR} =1.

The cause of an instruction TLB error interrupt can be found in the save/restore register 1 (SRR1). For bit assignments, refer to **Section 7.3.7.3.12 Implementation-Specific Instruction TLB Error Interrupt**. It is the software's responsibility to invoke the instruction storage interrupt handler.

11.7.4 Implementation-Specific Data TLB Error

The implementation-specific data TLB error interrupt occurs under one of the following conditions:

- The effective address of a **load**, **store**, **icbi**, **dcbz**, **dcbst**, **dcbf**, or **dcbi** instruction cannot be translated. Either the segment or page valid bit of this page is cleared in the translation table.
- The access violates storage protection.
- An attempt is made to write to a page with a negated change bit.

The data storage interrupt status register indicates the cause of the data TLB error interrupt. For bit assignments refer to **Section 7.3.7.3.14 Implementation-Specific Data TLB Error Interrupt**. It is the software's responsibility to invoke the data storage interrupt handler.

11.8 MANIPULATING THE TRANSLATION LOOKASIDE BUFFER

11.8.1 Reloading the Translation Lookaside Buffer

The TLB reload (tablewalk) function is performed in the software with optional hardware assistance in the following ways:

- Automatically stores the missed effective data or instruction address and default attributes in the MI_EPN or MD_EPN registers, respectively. This value is loaded into the selected entry on a write to MI_RPN or MD_RPN for the instruction and data translation lookaside buffer.
- Automatically updates the replacement location counter to point to the entry to be replaced. This value is placed in the xTLB_INDX field of the MI_CTR and MD_CTR.
- Generates a level one pointer when a **mfspr** Rx, M_TWB is performed by the concatenation of the level one table base and level one index. See Figure 11-2 and Figure 11-3 for details.
- Generates a level two pointer when a **mfspr** Rx, MD_TWC is performed by the concatenation of the level two table base and level two index.
- Performs a write to the TLB entry by loading the tablewalk level two entry value to the MI_RPN or MD_RPN register.
- A special register (M_TW) is available for the software tablewalk routine, in addition to the PowerPC architecture's special registers (SPRG0–SPRG3). Using this register allows for more efficient interrupt handling.

11.8.1.1 TRANSLATION RELOAD EXAMPLES. The following are code examples for generating the real page number using a two-level tree page table structure. The first example is for a data TLB reload and the second is for an instruction TLB reload. Notice that the following assumptions are made:

- M_TWB holds the base pointer to the first level table.
- Both instruction and data address translation is turned off (MSR_{IR} =0 and MSR_{DR} =0).

```
dtlb_swtw       mtspr   M_TW, R1        # save R1
                mfspr   R1, M_TWB       # load R1 with level one pointer
                lwz     R1, (R1)        # Load level one page entry
                mtspr   MD_TWC,R1       # save level two base pointer and
                                        # level one attributes
                mfspr   R1, MD_TWC      # load R1 with level two pointer
                                        # while taking into account the
                                        # page size
                lwz     R1, (R1)        # Load level two page entry
                mtspr   MD_RPN, R1      # Write TLB entry
                mfspr   R1, M_TW        # restore R1
                rfi


itlb_swtw       mtspr   M_TW, R1        # save R1
                mfspr   R1, SRR0        # load R1 with instruction miss
                                        # effective address (the same data
                                        # may be taken from the MI_EPN
                                        # register)
                mtspr   MD_EPN, R1      # save instruction miss effective
                                        # address in MD_EPN
                mfspr   R1, M_TWB       # load R1 with level one pointer
                lwz     R1, (R1)        # Load level one page entry
                mtspr   MI_TWC,R1       # save level one attributes
                mtspr   MD_TWC,R1       # save level two base pointer
                mfspr   R1, MD_TWC      # load R1 with level two pointer
                                        # while taking into account the
                                        # page size
                lwz     R1, (R1)        # Load level two page entry
                mtspr   MI_RPN, R1      # Write TLB entry
                mfspr   R1, M_TW        # restore R1
                rfi
```

11.8.2 Controlling the TLB Replacement Counter

The TLB replacement counter can be programmed to only select among the first six entries in each translation lookaside buffer by setting the $\overline{\text{RSV2I}}$ bit in the MI_CTR or the $\overline{\text{RSV2D}}$ bit in the MD_CTR. These control bits also affect the **tlbia** instruction. Replacement counters are cleared to zero after execution of the **tlbia** instruction and the counters decrement after an appropriate TLB reload.

11.8.3 Invalidating the Translation Lookaside Buffer

The MPC823 implements the **tlbie** instruction to invalidate the TLB entries. This instruction invalidates TLB entries in the translation lookaside buffer that hits, including the reserved entries. Notice that with 4K page size, the 22 most-significant bits of the effective address are used in the comparison because no segment registers are implemented. Although, for entries with larger page sizes than 4K, some of the lower bits of the effective page number are ignored. The ASID value in the entry is ignored for the purpose of matching an invalidated address, thus multiple entries can be invalidated if they have the same effective address and different ASID values.

The MPC823 supports the **tlbia** instruction to invalidate all entries in both translation lookaside buffers. If the $\overline{\text{RSV2D}}$ or $\overline{\text{RSV2I}}$ bit is set for a translation lookaside buffer, the two reserved entries will not be invalidated when **tlbia** is executed. However, the software can explicitly invalidate one or more of these entries by setting the xTLB_INDX field in the MD_CTR or MI_CTR, which negates the EV bit in the MD_EPN or MI_EPN register and performs a write to the appropriate MD_RPN or MI_RPN register. The translation lookaside buffers are not automatically invalidated on reset, but they are disabled. However, they must be invalidated under program control.

11.8.4 Loading the Reserved TLB Entries

To load a single reserved entry in the translation lookaside buffer, follow these steps:

1. Disable the translation lookaside buffer by clearing MSR_{IR} or MSR_{DR} as needed.
2. Clear the $\overline{\text{RSV2x}}$ bit in the Mx_CTR.
3. Invalidate the effective address of the reserved page by using the **tlbia** or **tlbie** instruction.
4. Set the xTLB_INDX fields of the Mx_CTR to the appropriate value (6 or 7).
5. Load the Mx_EPN register with the effective page number, the ASID with the reserved page, and set the EV bit to 1.
6. Run software tablewalk code to load the appropriate entry into the translation lookaside buffer. Refer to **Section 11.8.1.1 Translation Reload Examples** for examples of this code.
7. If needed, repeat the three previous steps to load other TLB entries.
8. Set the $\overline{\text{RSV2x}}$ bit in the Mx_CTR.

11.9 REQUIREMENTS FOR ACCESSING THE MEMORY MANAGEMENT UNIT CONTROL REGISTERS

All instruction and data memory management unit control registers should be accessed when instruction and data address translation is turned off. Prior to an **mtspr** MD_DBCAM Rx instruction, an **eieio** instruction should be placed and executed before you write to the Mx_CAM register.

SECTION 12
SYSTEM INTERFACE UNIT

The system interface unit controls system startup, initialization and operation, protection, as well as the external system bus. The system configuration and protection function controls the overall system and provides various monitors and timers, including the bus monitor, software watchdog timer, periodic interrupt timer, PowerPC™ decrementer, timebase, and real-time clock. The clock synthesizer generates the clock signals for other modules and external devices that the system interface unit uses. This circuitry generates the system clock from an inexpensive 32kHz crystal or an oscillator with a maximum frequency of 5MHz. The system interface unit supports various low-power modes that supply different ranges of power consumption, functionality, and wake-up time. The clock scheme supports low-power modes for applications that use baud rate generators and/or serial ports in standby mode. The main system clock can be changed dynamically, but the baud rate generators and serial ports work with a fixed frequency. For more information on clocks, refer to **Section 5 Clocks and Power Control**.

The external bus interface handles the transfer of information between internal buses and the memory or peripherals in the external address space. The MPC823 is designed to allow external bus masters to request and obtain mastership of the system bus. For additional information on bus operation, see **Section 13 External Bus Interface**. The memory controller module provides a glueless interface to many types of memory devices and peripherals and it supports a maximum of eight memory banks, each with their own device and timing attributes. Memory control services are provided to both internal and external masters. The MPC823 supports circuitboard test strategies through a user-accessible test logic that is fully compliant with information in **Section 21 IEEE 1149.1 Test Access Port**.

Note: The MPC823's external address bus is 26 bits wide, while the internal address bus is 32 bits wide. Therefore, external accesses are considered internally as 26-bit accesses (A[6:31]) with A[0:5] equal to 0, while internal accesses are full 32-bit accesses.

The PCMCIA host adapter module provides all control logic for a PCMCIA interface. A host adapter interface fully compliant with the PCMCIA Standard, Release 2.1+ (PC Card -16)). It can support one PCMCIA socket with a maximum of eight memory or I/O windows.

Note: Both the MPC823 and MPC821 have the same PCMCIA module except that the MPC823 has only one valid slot (Slot B). Programming a window to be assigned to Slot A may cause an erroneous operation.

12.1 FEATURES

The following is a list of the system interface unit's main features:

- System Configuration and Protection
- System Interrupt Configuration
- System Reset Monitoring and Generation
- Clock Synthesizer
- Power Management
- Real-Time Clock
- PowerPC Decrementer
- Timebase
- Periodic Interrupt Timer
- External Bus Interface Control
- Eight Memory Banks Supported By the Memory Controller
- Debug Support
- PCMCIA Host Adapter Module Supports Eight Memory or I/O Windows
- IEEE 1149.1 Test Access Port

12.2 SYSTEM CONFIGURATION AND PROTECTION

The MPC823 incorporates many system functions that normally must be provided in external circuits. It is designed to provide maximum system safeguards against hardware and/or software faults. The following features are provided in the system configuration and protection submodule:

- **Interrupt Configuration**—Allows you to configure the system according to your own requirements. The functions include control of parity checking, show cycle operation, and part and mask number constants.
- **Bus Monitor**—Monitors the $\overline{TA}$ response time for all bus accesses initiated by the internal masters. A $\overline{TEA}$ signal is asserted if the $\overline{TA}$ response limit is exceeded. The bus monitor measures time between $\overline{TS}$ and any termination of the bus cycle, including $\overline{TA}$, $\overline{TEA}$, and $\overline{RETRY}$.
- **Software Watchdog Timer**—Asserts a reset or nonmaskable interrupt that is selected by the system protection control register (SYPCR) if the software fails to service the software watchdog timer after a certain period of time. After system reset, the software watchdog timer, if enabled, selects a maximum timeout period and asserts a system reset if the timeout is reached. The software watchdog timer can be disabled or its timeout period can be changed in the SYPCR. Once the SYPCR is written, it cannot be written again until a system reset.

- **Periodic Interrupt Timer**—Generates periodic interrupts to be used with the real-time operating system or the application software. The periodic interrupt timer is clocked by the PITRTCLK clock, thus providing a period from 122 microseconds to 8,000 milliseconds assuming a 32.768kHz crystal. The periodic interrupt timer function can be disabled if it is not needed.
- **PowerPC Timebase Counter**—Provides a timebase reference for the operating system or application software. This 64-bit timebase counter is defined by the PowerPC architecture and has two independent reference registers that generate a maskable interrupt when the programmed value in one of the registers is reached. The associated bit in the timebase status register is set for the reference register that generated the interrupt. The timebase is clocked by the TMBCLK clock.
- **PowerPC Decrementer**—Provides a decrementer interrupt and is clocked at the same frequency as the timebase. This 32-bit decrementing counter is defined by the PowerPC architecture to be clocked by the TMBCLK clock. When it is driven by a 4MHz oscillator the period for the decrementer is 4,295 seconds, which is approximately 71.6 minutes.
- **Real-Time Clock**—Provides a time-of-day information to the operating system or application software. It is composed of a 45-bit counter and an alarm register. A maskable interrupt is generated when the counter reaches the value programmed in the alarm (RTCAL) register. The real-time clock is clocked by the PITRTCLK clock.
- **Freeze Support**—The system interface unit determines whether the software watchdog timer, periodic interrupt timer, timebase, decrementer, and real-time clock should continue to run in freeze mode.

Figure 12-1 illustrates a block diagram of the system configuration and protection logic.

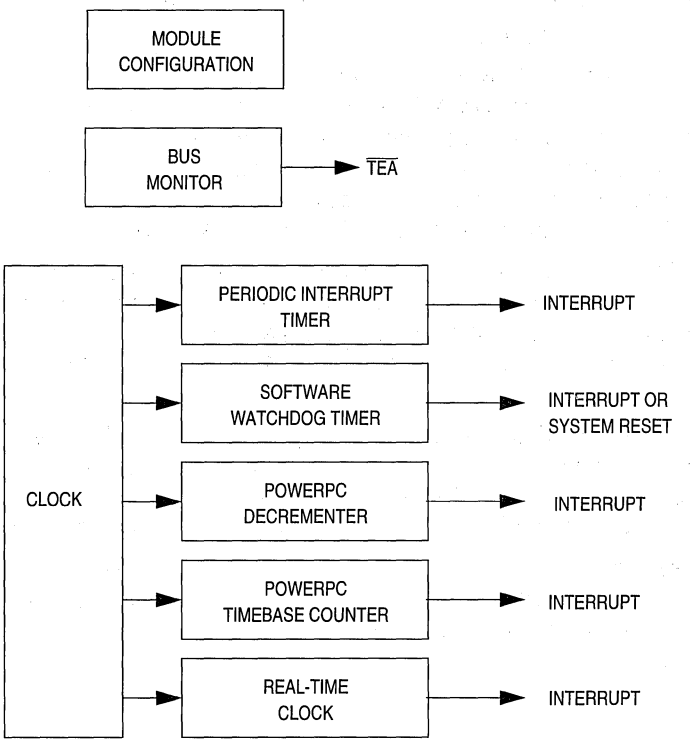


Figure 12-1. System Configuration and Protection Logic

12.3 INTERRUPT CONFIGURATION

Many aspects of MPC823 system configuration are controlled by the SIU module configuration register (SIUMCR). The SIUMCR primarily controls the external bus arbitration logic, external master support, and pin multiplexing. See **Section 12.12.1.1 SIU Module Configuration Register** for more information.

12.3.1 The Interrupt Structure

The system interface unit receives interrupts from internal sources, such as the periodic interrupt timer, real-time clock, communication processor module (CPM), and the external $\overline{IRQx}$ pins. The MPC823 interrupt structure is illustrated in Figure 12-2.

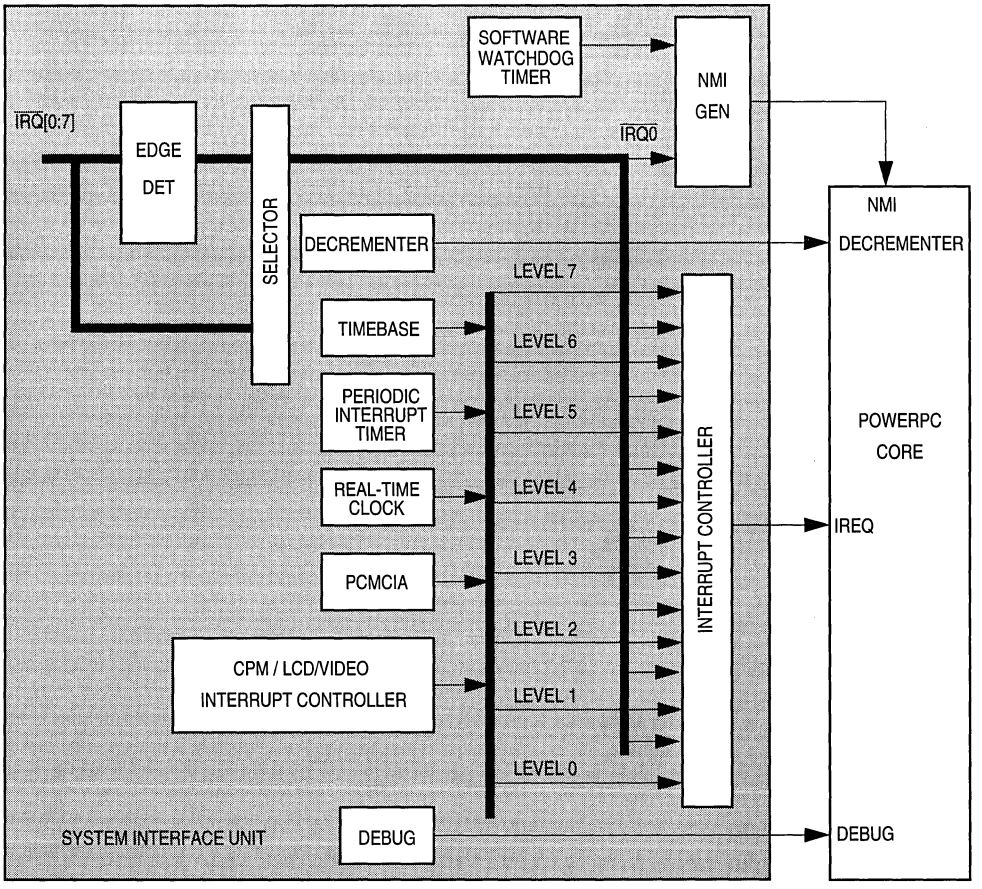


Figure 12-2. MPC823 Interrupt Structure

If it is programmed to generate an interrupt, the software watchdog timer always generates a nonmaskable interrupt (NMI) to the core. The external $\overline{\text{IRQ0}}$ pin can generate a nonmaskable interrupt as well. Notice that the core takes the system reset interrupt when a nonmaskable interrupt is asserted and the external interrupt when any other interrupt is asserted by the interrupt controller. Each one of the external $\overline{\text{IRQ}}$ pins has its own dedicated assigned priority level and there are eight additional interrupt priority levels. Each one of the system interface unit internal interrupt sources (the interrupt request that is generated by the communication processor module's interrupt controller) can be assigned by the software to any one of those eight interrupt priority levels. Thus, a very flexible interrupt scheme is realized.

12.3.2 Priority of the Interrupt Sources

The system interface unit has 15 interrupt sources that assert just one interrupt request to the core. There are seven external $\overline{\text{IRQ}}$ pins and eight interrupt levels. The priority between all interrupt sources is shown in Table 12-1.

Table 12-1. Priority of System Interface Unit Interrupt Sources

NUMBER	PRIORITY LEVEL	INTERRUPT SOURCE DESCRIPTION	INTERRUPT CODE
0	Highest	$\overline{\text{IRQ0}}$	00000000
1		Level 0	00000100
2		$\overline{\text{IRQ1}}$	00001000
3		Level 1	00001100
4		$\overline{\text{IRQ2}}$	00010000
5		Level 2	00010100
6		$\overline{\text{IRQ3}}$	00011000
7		Level 3	00011100
8		$\overline{\text{IRQ4}}$	00100000
9		Level 4	00100100
10		$\overline{\text{IRQ5}}$	00101000
11		Level 5	00101100
12		$\overline{\text{IRQ6}}$	00110000
13		Level 6	00110100
14		$\overline{\text{IRQ7}}$	00111000
15	Lowest	Level 7	00111100
16-31		Reserved	—

12.3.3 Programming the Interrupt Controller

The system interface unit's interrupt controller consists of the SIPEND, SIMASK, SIEL, and SIVEC registers.

12.3.3.1 SIU INTERRUPT PENDING REGISTER. The 32-bit SIU interrupt pending (SIPEND) register contains bits that individually correspond to an interrupt request. If they are set, the bits associated with internal exceptions indicate that an interrupt service is requested, if they are not masked by the corresponding bit in the SIMASK register. These bits reflect the status of the internal requesting device and they are cleared when the appropriate actions are software-initiated in the device itself. The bits associated with the $\overline{\text{IRQx}}$ pins have a different behavior depending on the sensitivity defined for them in the SIEL register. When an $\overline{\text{IRQx}}$ pin is defined as a "level" interrupt the corresponding bit behaves similar to the bits associated with internal interrupt sources. When an $\overline{\text{IRQx}}$ pin is defined as an "edge" interrupt and if the corresponding bit is set, it indicates that a falling edge was detected on the line. These bits are reset by writing a 1 to them.

SIPEND

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	IRQ0	LVL0	IRQ1	LVL1	IRQ2	LVL2	IRQ3	LVL3	IRQ4	LVL4	IRQ5	LVL5	IRQ6	LVL6	IRQ7	LV7
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x010															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x012															

IRQ—Interrupt Request 0–7

When set, this field indicates an edge-triggered interrupt of corresponding value.

0 = The appropriate interrupt is not pending.
1 = The appropriate interrupt is pending.

LVL—Level 0–7

When set, this field indicates a pending level interrupt of corresponding value.

0 = The appropriate interrupt is not pending.
1 = The appropriate interrupt is pending.

Bits 16–31—Reserved

These bits are reserved and should be set to 0.

12.3.3.2 SIU INTERRUPT MASK REGISTER. The 32-bit read/write SIU interrupt mask (SIMASK) register contains bits that individually correspond to the interrupt request bit in the SIPEND register. When a bit is set, it enables the generation of an interrupt request to the core. SIMASK is updated by the software and cleared at reset. It is the responsibility of the software to determine which of the interrupt sources are enabled at a given time.

SIMASK

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	IRM0	LVM0	IRM1	LVM1	IRM2	LVM2	IRM3	LVM3	IRM4	LVM4	IRM5	LVM5	IRM6	LVM6	IRM7	LVM7
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x014															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x016															

IRM—Interrupt Request Mask 0–7

When set, this field enables an $\overline{\text{IRQx}}$ interrupt request to be generated. The SIPEND register contains the IRQ bit. This bit does not mask the interrupt that goes to the CPU.

0 = Disable the generation of an IRQ bit in the SIPEND register.
1= Enable the generation of an IRQ bit in the SIPEND register.

LVM—Level Mask 0–7

When set, this field enables an interrupt request to be generated.

0 = Disable the generation of an IRQ bit in the SIPEND register.
1= Enable the generation of an IRQ bit in the SIPEND register.

Bits 16–31—Reserved

These bits are reserved and should be set to 0.

12.3.3.3 SIU INTERRUPT EDGE/LEVEL REGISTER. The 32-bit read/write SIU interrupt edge/level (SIEL) register contains pairs of bits that correspond to an external interrupt request. If set, the EDx bit specifies when a falling edge in the corresponding $\overline{IRQx}$ signal is an interrupt request. When the EDx bit is 0, a low logical level in the $\overline{IRQx}$ signal is an interrupt request. The WMx bit, if set, indicates that a low level detection in the corresponding interrupt request line causes the MPC823 to exit low-power mode.

SIEL

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	ED0	WM0	ED1	WM1	ED2	WM2	ED3	WM3	ED4	WM4	ED5	WM5	ED6	WM6	ED7	WM7
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x018															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x018															

ED—Edge Detect 0–7

0 = The bits specify that a low logical level in the $\overline{IRQx}$ signal is detected as an interrupt request.
1 = The bits specify that a falling edge in the corresponding $\overline{IRQx}$ signal is detected as an interrupt request.

WM—Wake-Up Mask 0–7

0 = Not allowed to exit from low-power mode.
1 = Allows low-level detection in the corresponding $\overline{IRQx}$ signal to exit or wake up the MPC823 from low-power mode.

Bits 16–31—Reserved

These bits are reserved and should be set to 0.

12.3.3.4 SIU INTERRUPT VECTOR REGISTER. The 32-bit read-only SIU interrupt vector (SIVEC) register contains an 8-bit code that represents the unmasked interrupt source of the highest priority level. The SIVEC register can be read as either a byte, half, or word. When read as a byte, a branch table can be used in which each entry contains one instruction (branch). When read as a half-word, each entry can contain a full routine of 256 instructions (max). The interrupt code is equal to the interrupt number multiplied by four, which allows indexing into the table. Refer to Figure 12-3 and Table 12-1 for details.

SIVEC

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	INTC								RESERVED							
RESET	0								0							
R/W	R								R							
ADDR	(IMMR & 0xFFFF0000) + 0x01C															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED															
RESET	0															
R/W	R															
ADDR	(IMMR & 0xFFFF0000) + 0x01E															

INTC—Interrupt Code

This field indicates the highest priority pending interrupt.

Bits 8–31—Reserved

These bits are reserved and should be set to 0. The value equals the interrupt number multiplied by four. See Table 12-1 for details.

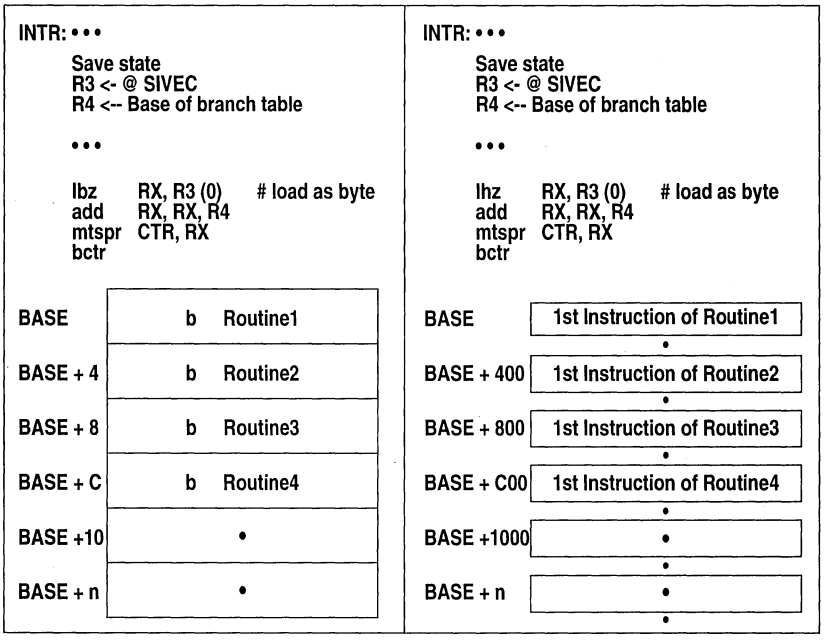


Figure 12-3. Interrupt Table Handling Example

12.4 THE BUS MONITOR

The bus monitor ensures that each bus cycle is terminated within a reasonable period of time. The MPC823 system interface unit provides a bus monitor option that monitors internally generated external bus accesses on the external bus. At the start of the transfer start ($\overline{TS}$) signal, the monitor begins counting and stops when the transfer acknowledge ($\overline{TA}$) or transfer error ($\overline{TEA}$) signal is asserted. Within the bus cycle, the burst cycle performs three functions—stops the previous count, resets, and restarts a new count until the next transfer acknowledge. The last transfer acknowledge of the burst cycle only stops the counter. When a $\overline{RETRY}$ signal is asserted, it has the same effect as the $\overline{TA}$ or $\overline{TEA}$ signal. If the monitor times out, a $\overline{TEA}$ signal is internally asserted by the bus monitor to terminate the cycle. The programmability of the timeout allows for a variation in system peripheral response time. The timing mechanism is clocked by the system clock divided by eight. The maximum value can be 2,040 system clocks. The bus monitor will always be active when the FRZ signal is asserted or when a debug mode request is pending, regardless of the state of the BMT bit.

Note: If the bus monitor is disabled, the transfer error conditions will not assert the $\overline{TEA}$ pin.

12.5 THE POWERPC DECREMENTER

The 32-bit decrementing counter is defined by the PowerPC architecture to provide a decrementer interrupt. This binary counter is clocked by the same frequency as the timebase. In the MPC823, the decrementer is clocked by the TMBCLK clock, so you must enable the TBE bit in the TBSCR for the decrementer to start.

$$T_{dec} = \frac{2^{32}}{(F_{tmbclk})}$$

The state of the decrementer is not affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ and, therefore, should be initialized by the software. The decrementer runs continuously after power-up. It continues counting while $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ are asserted and it is implemented with the following requirements in mind. The decrementer interrupt is also sent to the power-down wake-up logic, which allows the CPU to be awaken from power-down mode.

- The operation of the timebase and decrementer are coherent, which means the counters are driven by the same fundamental timebase.
- The decrementer is unaffected when read.
- When storing to the decrementer, the value in the decrementer is replaced with the value in the GPR.
- When Bit 0 (MSB) of the decrementer changes from 0 to 1, an interrupt request is signaled. If multiple decrementer interrupt requests are received before the first one is reported, only one interrupt is reported.
- If the decrementer is altered by the software and the content of Bit 0 is changed from 0 to 1, an interrupt request is signaled.

A decrementer exception causes a pending decrementer interrupt request in the core. When the decrementer interrupt is taken, the request is automatically cleared. The following chart shows some of the periods available for the decrementer, assuming that a 4MHz oscillator is used.

COUNT VALUE	TIMEOUT
0	1 microsecond
9	10. microseconds
99	100. microseconds
999	1.0 millisecond
9999	10.0 milliseconds

COUNT VALUE	TIMEOUT
999999	1.0 second
9999999	10.0 seconds
99999999	100.0 seconds
999999999	1000. seconds
FFFFFFFF(hex)	4295 seconds

12.5.1 Decrementer Register

The 32-bit decrementer (DEC) register is a special-purpose register defined by PowerPC architecture. The decrementer causes an interrupt whenever Bit 0 changes from a logic 0 to a logic 1. The contents of this register can be read or written to by the **mfspr** or **mtspr** instruction. This register is undefined at reset. The decrementer is powered by standby power and continues counting when standby power is applied. To enable the decrementer control bits, use the timebase control and status register. The decrementer and timebase share the same TMBCLK.

DEC

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	DEC															
RESET	—															
R/W	R/W															
SPR	22															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	DEC															
RESET	—															
R/W	R/W															
SPR	22															

NOTE: — = Undefined.

DEC—Decrementer

This field is used by a down counter to cause decrementer interrupts. A read of this register always returns the current count value from the down counter.

12.6 THE POWERPC TIMEBASE

The timebase is defined by the PowerPC architecture and is a 64-bit free-running binary counter that is incremented at a frequency determined by each implementation of the timebase. There is no interrupt or other indication generated when the count rolls over. The period of the timebase depends on the driving frequency. For the MPC823, the timebase is clocked by the TMBCLK clock and the period for the timebase is:

$$T_{TB} = \frac{2^{64}}{F_{tmbclk}}$$

The state of the timebase is unaffected by any resets and should be initialized by the software. Reads and writes of the timebase are restricted to special instructions. For the MPC823 implementation, it is not possible to read or write the entire timebase in a single instruction. Therefore, the **mttb** and **mftb** instructions are used to move the lower half of the timebase while the **mttbu** and **mftbu** instructions are used to move the upper half of the timebase. The timebase has two reference registers associated with it. A maskable interrupt is generated when the timebase count reaches the value programmed in one of the two reference registers and the two status bits indicate which of the two reference registers generated the interrupt.

12.6.1 Timebase Register

The special-purpose 64-bit timebase (TB) register contains a 64-bit integer that is periodically incremented. There is no automatic initialization of this register. The system software must perform the initialization. The contents of the register can be written by the **mtspr** instruction and read by the **mftb** or **mftbu** instruction.

TBU

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	TBU															
RESET	—															
R/W	R/W															
SPR	269 (READ), 285 (WRITE)															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	TBU															
RESET	—															
R/W	R/W															
SPR	269 (READ), 285 (WRITE)															

NOTE: — = Undefined.

TBU—Timebase Upper

The value stored in this field is used as an upper part of the timebase counter.

TBL

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	TBL															
RESET	—															
R/W	R/W															
SPR	268 (READ), 284 (WRITE)															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	TBL															
RESET																
R/W																
SPR	268 (READ), 284 (WRITE)															

NOTE: — = Undefined.

TBL—Timebase Lower

The value stored in this field is used as the lower part of the timebase register.

12.6.2 Timebase Reference Registers

There are two special-purpose 32-bit read/write timebase reference registers—TBREFU and TBREFL—associated with the upper and lower parts of the timebase. When there is a match between the contents of the timebase and the reference register, a reference interrupt is enabled in the timebase control and status register.

TBREFU

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	TBREFU															
RESET	—															
R/W	R/W															
SPR	204															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	TBREFU															
RESET	—															
R/W	R/W															
SPR	206															

NOTE: — = Undefined.

TBREFU—Timebase Reference Upper

These bits represent the 32-bit reference value for the upper part of the timebase.

TBREFL

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	TBREFL															
RESET	—															
R/W	R/W															
SPR	208															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	TBREFL															
RESET	—															
R/W	R/W															
SPR	20A															

NOTE: — = Undefined.

TBREFL—Timebase Reference Lower

These bits represent the 32-bit reference value for the lower part of the timebase.

12.6.3 Timebase Status and Control Register

The 16-bit read/write timebase status and control register (TBSCR) controls the timebase count enable and interrupt generation. It is also is used for reporting the source of the interrupts and can be read at any time. A status bit is cleared by writing a 1 (writing a zero has no effect) and more than one bit can be cleared at a time.

TBSCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	TBIRQ								REFA	REFB	RESERVED		REFAE	REFBE	TBF	TBE
RESET	0								0	0	0		0	0	0	0
R/W	R/W								R/W	R/W	R/W		R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x200															

TBIRQ—Timebase Interrupt Request

This field determines the interrupt priority level of the timebase. To specify a certain level, the appropriate bit should be set.

REFA and REFB—Reference Interrupt Status

If set, these bits indicate that a match has been detected between the corresponding reference register (TBREFU for REFA and TBREFL for REFB) and the timebase low register. The bit should be cleared by writing a 1.

Bits 10–11—Reserved

These bits are reserved and should be set to 0.

REFAE and REFBE—Reference Interrupt Enable

If one of these bits is asserted, the timebase generates an interrupt on assertion of the REFA or REFB bit. Otherwise, the interrupt is disabled.

TBF—Timebase Freeze Enable

0 = The timebase and decrementer are unaffected.
1 = The FRZ signal stops the timebase and decrementer.

TBE—Timebase Enable

0 = Disables timebase and decrementer operation.
1 = Enables timebase and decrementer operation.

12.7 THE REAL-TIME CLOCK

The real-time clock is a 45-bit counter that is clocked by the PITRTCLK clock. It is used to provide time-of-day indication to the operating system and application software. The counter is not affected by reset and operates in all low-power modes. It is initialized by the software. The real-time clock can be programmed to generate a maskable interrupt when the time value matches the value programmed in the associated alarm register. It can also be programmed to generate an interrupt once every second. A control and status register is used to enable or disable the different functions and report the interrupt source. The real-time clock registers—RTCSC, RTC, RTSEC, and RTCAL—are protected ("locked") from accidental writes after $\overline{\text{PORESET}}$. To unlock the registers, you must write a key word (55CCAA33) to the RTSCK register. Refer to **Section 5.4.2 Keep-Alive Power** for more information.

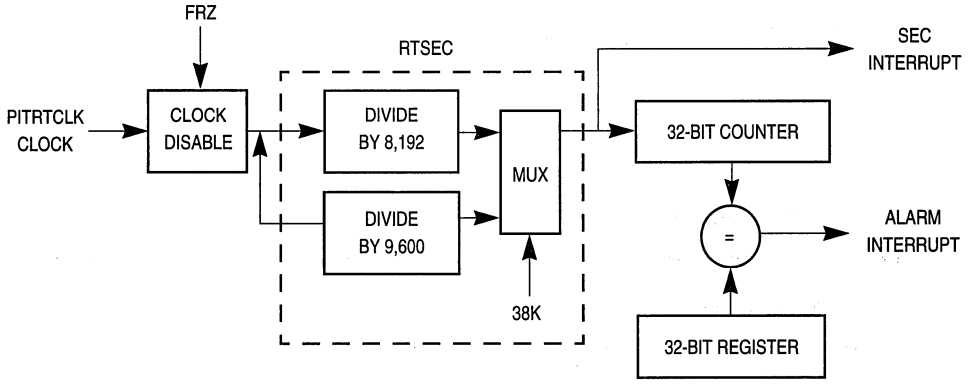


Figure 12-4. Real-Time Clock Block Diagram

12.7.1 Real-Time Clock Status and Control Register

The real-time clock status and control (RTCSC) register is used to enable the different real-time clock functions and for reporting the source of the interrupts. A status bit is cleared by writing a 1 (writing a zero has no effect) and more than one status bit can be cleared at a time. This register can be read at any time.

RTCSC

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RTCIRQ								SEC	ALR	RES	38K	SIE	ALE	RTF	RTE
RESET	0								0	0	0	—	0	0	0	—
R/W	R/W								R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x220															

NOTE: — = Undefined.

RTCIRQ—Real-Time Clock Interrupt Request

This field controls the real-time clock's interrupt priority level.

SEC—Once Per Second Interrupt

This status bit is set every second and should be cleared by the software.

ALR—Alarm Interrupt

This status bit is set when the value of the real-time clock is equal to the value programmed in the RTCAL register.

Bit 10—Reserved

This bit is reserved and should be set to 0.

38K—Real-Time Clock Source Select

The software must set this bit to get the proper timing of a second.

0 = Assumes that it is driven by a 32.768kHz crystal
1 = Assumes that it is driven by a 38.4kHz crystal.

SIE—Seconds Interrupt Enable

This bit allows the real-time clock to generate an interrupt when the SEC bit is set.

0 = Disables seconds interrupt.
1 = The real-time clock generates an interrupt.

ALE—Alarm Interrupt Enable

This bit allows the real-time clock to generate an interrupt when the ALR bit is set.

0 = Disables the seconds interrupt.
1 = The real-time clock generates an interrupt.

RTF—Real-Time Clock Freeze Enable

0 = The real-time clock is unaffected by the FRZ signal.
1 = The FRZ signal stops the real-time clock.

RTE—Real-Time Clock Enable

0 = The real-time clock timers are disabled.
1 = The real-time clock timers are enabled.

12.7.2 Real-Time Clock Register

The 32-bit real-time clock (RTC) register contains the current value of the real-time clock. The maximum value is approximately 136 years.

RTC

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RTC															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x224															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RTC															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x226															

NOTE: — = Undefined.

RTC—Real-Time Clock

This field represents time measured in seconds. Each unit represents one second.

12.7.3 Real-Time Clock Alarm Seconds Register

The 32-bit real-time clock alarm seconds (RTSEC) register contains the value to be compared with the RTC register and will generate an alarm condition if they match. The maximum value is approximately 136 years.

RTSEC

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	ASEC															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x228															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	ASEC															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x22A															

NOTE: — = Undefined.

ASEC—Alarm Seconds

This field contains the value to be compared with the value in the RTC register.

12.7.4 Real-Time Clock Alarm Register

The 32-bit read/write real-time clock alarm (RTCAL) register is an alarm reference register. When the RTC register increments to the value stored in this register, an alarm interrupt is generated.

RTCAL

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	ALARM															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x22C															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	ALARM															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x22E															

NOTE: — = Undefined.

ALARM—Alarm Reference Counter

This field indicates that an alarm interrupt will be generated as soon as there is a match between this field and the corresponding bits in the RTC register. The alarm has a 1 second resolution.

12.8 THE PERIODIC INTERRUPT TIMER

The periodic interrupt timer consists of a 16-bit counter clocked by a PITRTCLK clock supplied by the clock module. It decrements to zero when loaded with a value from the periodic interrupt timer count register (PITC) and after the timer reaches zero, the PS bit is set and an interrupt is generated if the PIE bit is a logic 1. At the next input clock edge, the value in the PITC register is loaded into the counter and the process starts all over again. When a new value is loaded into the PITC register, the periodic interrupt timer is updated, the divider is reset, and the counter starts counting. If the PS bit is not cleared, it generates an interrupt at the interrupt controller and the interrupt remains pending until it is cleared. If the PS bit is set again, prior to being cleared, the interrupt remains pending until the PS bit is cleared. Any write to the PITC register stops the current countdown and the count resumes with a new value in the PITC. If the PTE bit is not set, the periodic interrupt timer is unable to count and retains the old count value. Reads of the periodic interrupt timer have no effect on it.

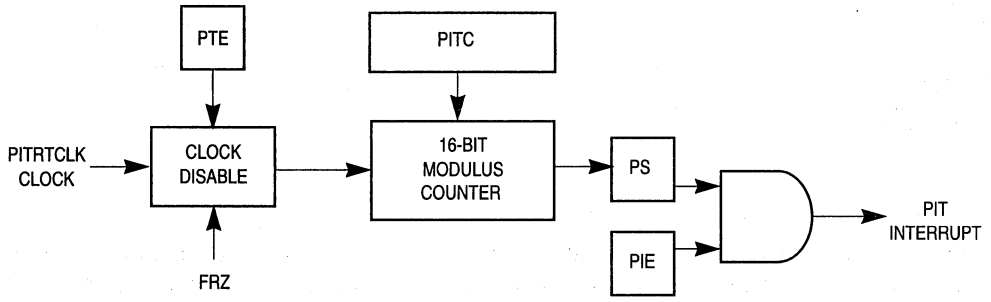


Figure 12-5. Periodic Interrupt Timer Block Diagram

The timeout period is calculated as:

$$PIT_{period} = \frac{PITC + 1}{F_{pitrtclk}} = \frac{PITC + 1}{\left(\frac{ExternalClock}{1or128}\right) \div 4}$$

Solving this equation using a 32.768kHz external clock gives:

$$PITperiod = \frac{PITC + 1}{8192}$$

This gives a range from 122 microseconds with a PITC of 0x0000 to a maximum of 8 seconds with a PITC of 0xFFFF.

12.8.1 Periodic Interrupt Status and Control Register

The read/write periodic interrupt status and control register (PISCR) contains the interrupt request level and the interrupt status bits. It also controls the 16 bits to be loaded in a modulus counter.

PISCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PIRQ								PS	RESERVED				PIE	PITF	PTE
RESET	0								0	0				0	0	1
R/W	R/W								R/W	R/W				R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x240															

PIRQ—Periodic Interrupt Request Level

This field allows you to configure any interrupt level for periodic interrupts. See Figure 12-2 for interrupt request levels.

PS—Periodic Interrupt Status

This bit can be negated by writing a 1 to it (zero has no effect).

0 = The periodic interrupt timer is unaffected.
1 = The periodic interrupt timer has issued an interrupt.

Bits 9–12—Reserved

These bits are reserved and should be set to 0.

PIE—Periodic Interrupt Enable

0 = Disables the PS bit.
1 = Enables the PS bit to generate an interrupt.

PITF—Periodic Interrupt Timer Freeze Enable

0 = The periodic interrupt timer is unaffected by the FRZ signal.
1 = The FRZ signal stops the periodic interrupt timer.

PTE—Periodic Timer Enable

0 = The periodic interrupt timer is disabled.
1 = The periodic interrupt timer is enabled.

12.8.2 Periodic Interrupt Timer Count Register

The read/write periodic interrupt timer count (PITC) register contains a 16-bit value that will be loaded into the periodic interrupt down counter.

PITC

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PITC															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x244															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x246															

NOTE: — = Undefined.

PITC—Periodic Interrupt Timer Count

This field contains the count for the periodic timer. If this field is loaded with the value 0xFFFF, the maximum count period will be selected.

Bits 16–31—Reserved

These bits are reserved and should be set to 0.

12.8.3 Periodic Interrupt Timer Register

The periodic interrupt timer register (PITR) is a read-only register that shows the current value in the periodic interrupt down counter. Writes to this register do not affect this register and reads of this register do not have any affect on the counter.

PITR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PIT															
RESET	—															
R/W	R															
ADDR	(IMMR & 0xFFFF0000) + 0x248															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED															
RESET	0															
R/W	R															
ADDR	(IMMR & 0xFFFF0000) + 0x24A															

NOTE: — = Undefined.

PIT—Periodic Interrupt Timing Count

This field contains the current count remaining for the periodic timer. Writes have no effect on this field.

Bits 16–31—Reserved

These bits are reserved and should be set to 0.

12.9 THE SOFTWARE WATCHDOG TIMER

The system interface unit provides the software watchdog timer (SWT) option that prevents system lockout when the software gets trapped in loops without a controlled exit. The software watchdog timer is enabled after system reset to automatically generate a system reset if it times out. If you do not need the software watchdog timer, you must clear the SWE bit in the system protection control register (SYPCR) to disable it. If it is used, the software watchdog timer requires a special service sequence to be executed on a periodic basis. If this periodic servicing action does not occur, the software watchdog timer times out and issues a reset or a nonmaskable interrupt, which is programmed in the SWRI bit of the SYPCR. Once the SYPCR register is written by the software, the state of the SWE bit cannot be changed. Refer to the system configuration and protection registers for more information. To service the software watchdog timer, write 0x556C and 0xAA39 to the software service register.

This sequence clears the watchdog timer and the timing process begins again. If any value other than 0x556C or 0xAA39 is written to the software service register (SWSR), the entire sequence must start over. Although the writes must occur in the correct order before a timeout occurs, any number of instructions may be executed between the writes. This allows interrupts and exceptions to occur between the two writes when necessary. Refer to Figure 12-6 for more information.

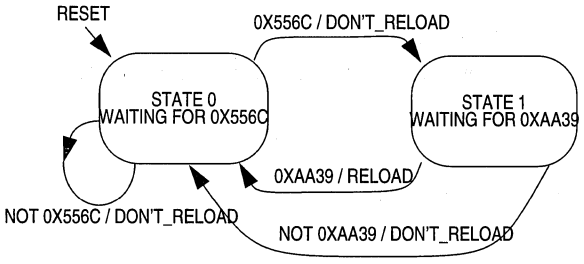


Figure 12-6. Software Watchdog Timer Service State Diagram

Although most software disciplines permit or encourage the watchdog concept, some systems require a selection of timeout periods. For this reason, the software watchdog timer must provide a selectable range for the timeout period. Figure 12-7 illustrates the present method for handling this requirement. Figure 12-7 also shows the range that the value in the SWTC field determines. This value is then loaded into a 16-bit decrementer clocked by the system clock. When necessary, an additional divide by 2,048 prescaler is used.

The decrementer begins counting when it is loaded with a value from the SWTC field. After the timer reaches 0x0, a software watchdog expiration request is issued to the reset or NMI control logic. At reset, the value in the SWTC register is set to the maximum value and is loaded into the software watchdog register (SWR) again, thus starting the process over. When a new value is loaded into the SWTC register, the software watchdog timer will not be updated until the servicing sequence is written to the SWSR register. If the SWE bit is loaded with the value 0, the modulus counter will not count.

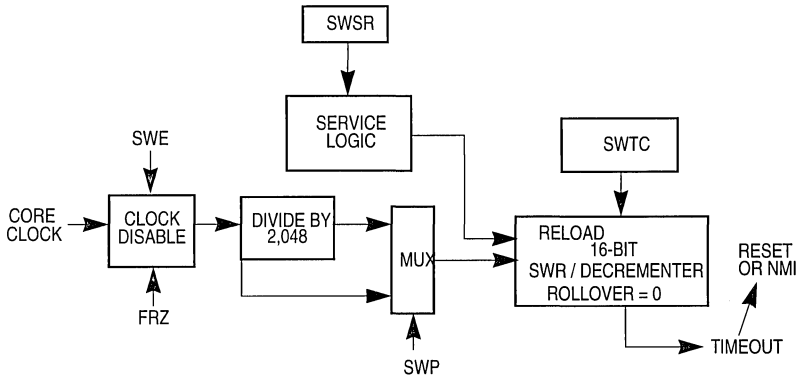


Figure 12-7. Software Watchdog Timer Block Diagram

12.9.1 Software Service Register

The software service register (SWSR) is the location that the software watchdog timer servicing sequence writes to. To prevent a SWT timeout, a write of 0x556C followed by 0xAA39 should be written to this register. The SWSR can be written at any time, but returns all zeros when read.

SWSR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	SEQ															
RESET	0															
R/W	W															
ADDR	(IMMR & 0xFFFF0000) + 0x00E															

SEQ—Sequence

This field is the pattern that is used to control the state of the software watchdog timer.

12.10 FREEZE OPERATION

When the FRZ signal is asserted, the clocks to the software watchdog, periodic interrupt timer, real-time clock, timebase counter, and decrementer can be disabled. This is controlled by the associated bits in the control register of each timer. If they are programmed to stop counting when FRZ is asserted, the counters maintain their values until FRZ is negated. The bus monitor, however, will be enabled regardless of this signal's state.

12.10.1 Low-Power Stop Operation

When the PowerPC core is set in a low-power mode (doze, sleep, deep sleep), the software watchdog timer is frozen. It remains frozen and maintains its count value until the core exits this mode and continues to execute instructions. The periodic interrupt timer, decrementer, and timebase are not influenced by these low-power modes and they continue to run at their respective frequencies. These timers can generate an interrupt to bring the MPC823 out of the low-power modes.

12.11 MULTIPLEXING THE SYSTEM INTERFACE UNIT PINS

Due to the limited number of pins available in the MPC823 package, some of the functionalities share pins. The actual MPC823 pinout is illustrated in **Section 2 External Signals**. The following table shows how the functionality is controlled on each pin.

Table 12-2. Multiplexing Control

PIN NAME	PIN CONFIGURATION CONTROL
TSIZ0/REG	Dynamically active depending if the transaction addresses a slave controlled by the PCMCIA interface.
BDIP/GPL_B5 RSV/IRQ2 KR/RETRY/IRQ4/SPKROUT DP[0:3]/IRQ[3:6] FRZ/IRQ6	Programmed in the SIUMCR.
CS6/CE1_B CS7/CE2_B	Address matching and bank valid bits. When there is a transfer such that there is a match in either memory controller bank 6 or any PCMCIA bank mapped to slot B, the CS(6)/CE(1)_B will be asserted. When there is a transfer such that there is a match in either memory controller bank 7 or any PCMCIA bank mapped to slot B, the CS(7)/CE(2)_B will be asserted.
WE0/BS_AB0/IORD WE1/BS_AB1/IOWR WE2/BS_AB2/PCOE WE3/BS_AB3/PCWE	Dynamically active depending on the machine (GPCM, UPMB, or PCMCIA interface) assigned to control the required slave.
GPL_A0/GPL_B0	Dynamically active depending on the machine (UPMA or UPMB) assigned to control the required slave.
OE/GPL_A1/GPL_B1	Dynamically active depending on the machine (GPCM, UPMA, or UPMB) assigned to control the required slave.
GPL_A[2:3]/GPL_B[2:3]/ CS[2:3]	GPL_A(2:3)/GPL_B(2:3): Dynamically active depending on the machine (UPMA or UPMB) assigned to control the required slave. GPL_A(2:3)/CS(2:3): Programmed in the SIUMCR.
ALE_B/DSCK/AT1 IP_B[0:1]/IWP[0:1]/VFLS[0:1] IP_B2/IOIS16_B/AT2 IP_B3/IWP2/VF2 IP_B4/LWP0/VF0 IP_B5/LWP1/VF1 IP_B6/DSDI/AT0 IP_B7/PTR/AT3 TDI/DSDI TCK/DSCK TDO/DSDO	Programmed in the SIUMCR and hard reset configuration.
UPWAITA/GPL_A4/AS	Programmed in the SIUMCR and MAMR of the memory controller.
OP2/MODCK2/STS OP3/MODCK2/DSDO	At power-on reset, this functions as MODCK[1:2] Otherwise, programmed in the SIUMCR and hard reset configuration.

12.12 PROGRAMMING THE SYSTEM INTERFACE UNIT

12.12.1 System Configuration and Protection Registers

12.12.1.1 SIU MODULE CONFIGURATION REGISTER. The SIU module configuration register (SIUMCR) contains bits that configure various features in the system interface unit.

SIUMCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	EARB	EARP			RESERVED				DSHW	DBGC		DBPC		RES	FRC	DLK
RESET	0	0			0				0	0		0		0	0	0
R/W	R/W	R/W			R/W				R/W	R/W		R/W		R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x000															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	OPAR	PNCS	DPC	MPRE	MLRC		AEME	SEME	RES	GB5E	B2DD	B3DD	RESERVED			
RESET	0	0	0	0	0		0	0	0	0	0	0	0			
R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W			
ADDR	(IMMR & 0xFFFF0000) + 0x002															

NOTE: The OPAR and PNCS bits swapped places and are now different from the MPC821.

EARB—External Arbitration

If the EARB bit is set, then external arbitration is assumed. If it is cleared, internal arbitration is performed. For more information, see **Section 13.4.6 Arbitration Phase-Related Signals**.

EARP—External Arbitration Request Priority

This field defines the priority of the external master's arbitration request. This field is valid when EARB is cleared. 000 is the lowest priority level and 111 the highest. For more information, refer to Figure 13-20 in **Section 13 External Bus Interface.**

Bits 4–7, 13, 24, and 28–31—Reserved

These bits are reserved and should be set to 0.

DSHW—Data Show Cycles

This bit selects the show cycle mode to be applied to data cycles. Instruction show cycles are programmed in the ICTRL register. The following chart shows the meaning of the bit. Refer to **Section 20.6.2 Development Port Registers** for more information. This bit is locked by the DLK bit.

0 = Disable show cycles for all internal data cycles.
1 = Show address and data of all internal data cycles.

DBGC—Debug Pin Configuration

This field configures the debug pin functionality. The default value is set by the hard reset configuration word, as shown in **Section 4.3.1.1 Hard Reset Configuration Word.**

00 = IP_B(0:1)/IWP(0:1)/VFLS[0:1] functions as IP_B[0:1].
IP_B3/IWP2/VF2 functions as IP_B3.
IP_B4/LWP0/VF0 functions as IP_B4.
IP_B5/LWP1/VF1 functions as P_B5.
OP2/MODCK1/$\overline{\text{STS}}$ functions as OP2.
ALE_B/DSCK/AT1 functions as ALE_B.
IP_B2/AT2 functions as IP_B2.
IP_B6/DSDI/AT0 functions as IP_B6.
IP_B7/$\overline{\text{PTR}}$/AT3 functions as IP_B7.
OP3/MODCK2/DSDO functions as OP3.

01 = IP_B[0:1]/IWP(0:1)/VFLS[0:1] functions as IWP[0:1].
IP_B3/IWP2/VF2 functions as IWP2.
IP_B4/LWP0/VF0 functions as LWP0.
IP_B5/LWP1/VF1 functions as LWP1.
OP2/MODCK1/$\overline{\text{STS}}$ functions as $\overline{\text{STS}}$.
ALE_B/DSCK/AT1 functions as AT1.
IP_B2/AT2 functions as AT2.
IP_B6/DSDI/AT0 functions as AT0.
IP_B7/$\overline{\text{PTR}}$/AT3 functions as AT3.
OP3/MODCK2/DSDO functions as OP3.

10 = Reserved.

11 = IP_B[0:1]/IWP[0:1]/VFLS[0:1] functions as VFLS[0:1].
IP_B3/IWP2/VF2 functions as VF2.
IP_B4/LWP0/VF0 functions as VF0.
IP_B5/LWP1/VF1 functions as VF1.
OP2/MODCK1/$\overline{\text{STS}}$ functions as $\overline{\text{STS}}$.
ALE_B/DSCK/AT1 functions as AT1.
IP_B2/AT2 functions as AT2.
IP_B6/DSDI/AT0 functions as AT0.
IP_B7/$\overline{\text{PTR}}$/AT3 functions as AT3.
OP3/MODCK2/DSDO functions as OP3.

DBPC—Debug Port Pins Configuration

This field determines the active pins for the development port. The default value is set by the hard reset configuration word, as shown in **Section 4.3.1.1 Hard Reset Configuration Word**.

00 = ALE_B/DSCK/AT1 functions as defined by DBGC.
IP_B6/DSDI/AT0 functions as defined by DBGC.
OP3/MODCK2/DSDO functions as defined by DBGC.
IP_B7/$\overline{\text{PTR}}$/AT3 functions as defined by DBGC.
TCK/DSCK functions as DSCK.

TDI/DSDI functions as DSDI.
TDO/DSDO functions as DSDO.

01 = ALE_B/DSCK/AT1 functions as defined by DBGC.
IP_B6/DSDI/AT0 functions as defined by DBGC.
OP3/MODCK2/DSDO functions as defined by DBGC.
IP_B7/$\overline{\text{PTR}}$/AT3 functions as defined by DBGC.
TCK/DSCK functions as TCK.
TDI/DSDI functions as TDI.
TDO/DSDO functions as TDO.

10 = Reserved.

11 = ALE_B/DSCK/AT1 functions as DSCK.
IP_B6/DSDI/AT0 functions as DSDI.
OP3/MODCK2/DSDO functions as OP3.
IP_B7/$\overline{\text{PTR}}$/AT3 functions as $\overline{\text{PTR}}$.
TCK/DSCK functions as TCK.
TDI/DSDI functions as TDI.
TDO/DSDO functions as TDO.

FRC—FRZ Pin Configuration

This bit configures the functionality of the FRZ/$\overline{\text{IRQ6}}$ pin.

0 = FRZ/$\overline{\text{IRQ6}}$ functions as FRZ.
1 = FRZ/$\overline{\text{IRQ6}}$ functions as $\overline{\text{IRQ6}}$.

DLK—Debug Register Lock

If this bit is set, bits 8–15 are locked and writes to those bits are no longer performed. These bits are writable in test mode once the internal FRZ signal is asserted, regardless of the state of DLK. This bit is cleared by reset.

OPAR—Odd Parity

This bit is used to program odd or even parity. It can also be used to generate parity errors for testing purposes by writing the memory with OPAR = 1 and reading the memory with OPAR = 0.

PNCS—Parity Enable For Nonmemory Controller Regions

This bit enables parity generation/checking for memory regions not controlled by the MPC823 memory controller.

DPC—Data Parity Pins Configuration

This bit configures the functionality of the DP[0:3]/$\overline{\text{IRQ}}$[3:6] pins.

0 = DP[0:3]/IRQ[3:6] functions as IRQ[3:6].
1 = DP[0:3]/IRQ[3:6] functions as DP[0:3].

MPRE—Multiprocessors Reservation Enable

If this bit is set, then the interprocessor reservation protocol is enabled. The RSV pin functions as defined in **Section 13.4.10 Storage Reservation Protocol**.

0 = RSV/IRQ2 functions as IRQ2.
1= RSV/IRQ2 functions as RSV.

MLRC—Multi-Level Reservation Control

This field configures the functionality of the KR/RETRY/IRQ4/SPKROUT pins.

00 = KR/RETRY/IRQ4/SPKROUT functions as IRQ4.
01 = KR/RETRY/IRQ4/SPKROUT is three-stated.
10 = KR/RETRY/IRQ4/SPKROUT functions as KR/RETRY.
11 = KR/RETRY/IRQ4/SPKROUT functions as SPKROUT.

AEME—Asynchronous External Master Enable

This bit configures how the memory controller refers to external asynchronous masters initiating a transaction. If this bit is set, the memory controller interprets any assertion on the AS pin as an external asynchronous master initiating a transaction. If it is reset, the memory controller ignores the value of the AS pin. This bit and the GPLA4DIS bit of the machine A mode register controls the direction and functionality of the UPWAITA/GPL_A4/AS pins.

AEME	GPLA4DIS	PIN USAGE	UPWAIT VALUE	AS VALUE
X	0	Functions as GPL_A4	GND	VCC
0	1	Functions as UPWAIT	Pin value	VCC
1	1	Functions as AS	GND	Pin value

SEME—Synchronous External Master Enable

This bit configures how the memory controller refers to external synchronous masters initiating a transaction. If this bit is set, the memory controller interprets any assertion on the TS pin the external bus does not own as an external synchronous master initiating a transaction. If it is reset, the memory controller ignores the value of the TS pin when it does not own the external bus. When the MPC823 owns the bus, the memory interprets the assertion of the TS pin as an internal request.

GB5E—GPL_B5 Enable

0 = The BDIP functionality is active.
1 = The GPL_B5 of the memory controller functionality is active.

B2DD—Bank 2 Double Drive

If this bit is set, the CS2 signal is reflected on GPL_x2.

B3DD—Bank 3 Double Drive

If this bit is set, the CS3 signal is reflected on GPL_x3.

12.12.1.2 INTERNAL MEMORY MAP REGISTER. The internal memory map register (IMMR) is located within the PowerPC special register space. It contains the identification of a specific device, as well as a base for the internal memory map. Based on the value read from this register, the software can deduce the availability and location of any on-chip system resource. The contents of this register can be read by the **mfspr** instruction and the ISB field can be written by the **mtspr** instruction. However, the PARTNUM and MASKNUM fields are mask programmed and cannot be changed for any device.

IMMR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	ISB															
RESET	1								0							
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x638															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	PARTNUM								MASKNUM							
RESET	0	0	1	0	0	0	0	0	*							
R/W	R								R							
ADDR	(IMMR & 0xFFFF0000) + 0x638															

NOTE: * The value depends on the mask revision.

ISB—Internal Space Base

This read/write field defines the base address of the internal memory space. The initial value of this field can be configured at reset to one of four addresses and changed to any value by the software. The number of programmable bits in this field and the resolution of the location of internal space depends on the internal memory space of the specific implementation. In the MPC823, you can program all 16 bits. For the information on the device's internal memory map, refer to **Section 3 Memory Map** and for the available default initial values refer to **Section 4.3.1.1 Hard Reset Configuration Word**.

PARTNUM—Part Number

This read-only field is mask programmed with a code corresponding to the part number of the part on which the system interface unit is located. It is intended to help with factory test and user code that is sensitive to part refinements. This field changes as the part number changes. For example, it would change if a new module is added or if the size of the memory module is revised. However, it would not change if the part is revised to fix a bug in an existing module. The MPC823 has a part number of 0x20. The other byte of information reflects the revision number. Refer to our website for the corresponding revision number for your particular version of the silicon.

MASKNUM—Mask Number

This read-only field is mask-programmed with a code corresponding to the mask number of the part on which the system interface unit is located. It is intended to help with factory test and user code that is sensitive to part refinements. As a result, the value of this field depends on the mask revision.

12.12.1.3 SYSTEM PROTECTION CONTROL REGISTER. The system protection control register (SYPCR) controls the system monitors, software watchdog period, and bus monitor timing. This register can be read at any time, but can only be written once after system reset.

SYPCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	SWTC															
RESET	1															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x004															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	BMT								BME	RESERVED			SWF	SWE	SWRI	SWP
RESET	1								0	0			0	1	1	1
R/W	R/W								R/W	R/W			R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x006															

SWTC—Software Watchdog Timer Count

This field contains the count value for the software watchdog timer.

BME—Bus Monitor Enable

This bit controls the operation of the bus monitor when an internal to external bus cycle is executed.

0 = Disable the bus monitor.
1 = Enable the bus monitor.

Note: If the bus monitor is disabled, the transfer error conditions will not assert the $\overline{\text{TEA}}$ pin.

BMT—Bus Monitor Timing

This field defines the timeout period, in 8 system clock resolution, for the bus monitor. The maximum timeout is 2,040 clocks.

Bits 25–27—Reserved

These bits are reserved and should be set to 0.

SWF—Software Watchdog Freeze

0 = The software watchdog timer continues counting even if the FRZ signal is asserted.
1 = The software watchdog timer stops counting when the FRZ signal is asserted.

SWE—Software Watchdog Enable

This bit enables the software watchdog timer. To disable the software watchdog timer, it should be cleared by the software after a system reset.

SWRI—Software Watchdog Reset/Interrupt Select

0 = The software watchdog timer causes a nonmaskable interrupt to the core.
1 = The software watchdog timer causes a system reset (default).

SWP—Software Watchdog Prescale

0 = The software watchdog timer is not prescaled.
1 = The software watchdog timer is prescaled by a factor of 2,048.

12.12.1.4 TRANSFER ERROR STATUS REGISTER. The transfer error status register (TESR) contains a bit for each exception source generated by a transfer error. A bit set to logic 1 indicates what type of transfer error exception occurred since the last time the bits were cleared. The bits are cleared by reset or by writing a 1 to the appropriate bit. Canceled speculative accesses that do not cause an interrupt may set these bits. The register has two identical sets of fields–one is associated with instruction transfers and the other with data transfers.

TESR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
RESET	0															
R/W	R															
ADDR	(IMMR & 0xFFFF0000) + 0x020															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED		IEXT	ITMT	IPB0	IPB1	IPB2	IPB3	RESERVED		DEXT	DTMT	DPB0	DPB1	DPB2	DPB3
RESET	0		0	0	0	0	0	0	0		0	0	0	0	0	0
R/W	R		R	R	R	R	R	R	R		R	R	R	R	R	R
ADDR	(IMMR & 0xFFFF0000) + 0x022															

Bits 0–17 and 24–25—Reserved

These bits are reserved and should be set to 0.

IEXT—Instruction External Transfer Error Acknowledge

This bit is set if the cycle is terminated by an externally generated $\overline{\text{TEA}}$ signal when an instruction fetch is initiated.

ITMT—Instruction Transfer Monitor Timeout

This bit is set if the cycle is terminated by a bus monitor timeout when an instruction fetch is initiated.

IPB0–IPB3—Instruction Parity Error on Bytes 0–3

There are four parity error status bits for each byte lane. One of these is set for the byte that had a parity error when an instruction was fetched. Parity check for a memory region that is not controlled by the memory controller is enabled by the PNCS bit in the SIUMCR, as shown in **Section 12.12.1.1 SIU Module Configuration Register**.

DEXT—Data External Transfer Error Acknowledge

This bit is set if the cycle is terminated by an externally generated $\overline{\text{TEA}}$ signal when a data load or store is requested by an internal master.

DTMT—Data Transfer Monitor Timeout

This bit is set if the cycle is terminated by a bus monitor timeout when a data load or store is requested by an internal master.

DPB0–DPB3—Data Parity Error On Bytes 0–3

There are four parity error status bits for each byte lane. One of these is set for the byte that had a parity error when a data load was requested by an internal master. Parity check for a memory region that is not controlled by the memory controller is enabled by the PNCS bit in the SIUMCR, as shown in **Section 12.12.1.1 SIU Module Configuration Register**.

SECTION 13
EXTERNAL BUS INTERFACE

The MPC823 bus is synchronous and burstable. Signals driven on this bus are required to make the setup and hold time relative to the bus clock's rising edge. This bus has the ability to support multiple masters. The MPC823 architecture supports byte, half-word, and word operands allowing access to 8-,16-, and 32-bit data ports through the use of synchronous cycles controlled by the transfer size output (TSIZx) signals. The slave access to 16- and 8-bit ports is controlled by the memory controller.

13.1 FEATURES

The following is a list of the bus interface's main features:

- 26-Bit Address Bus with Transfer Size Indication
- 32-Bit Data Bus
- TTL-Compatible Interface
- Internal On-Chip Bus Arbitration Logic Supports One External Bus Master
- Chip-Select and Wait State Generation
- Supports Many Different Memory Types
- Asynchronous DRAM Interface Support
- Flash ROM Programming Support
- Compatible with PowerPC Architecture
- Interfaces to Slave Devices Easily
- Synchronous Bus Operation
- Data Parity Support

13.2 TRANSFER SIGNALS

The bus transfers information between the MPC823 and the external memory or peripheral device. External devices can accept or provide 8,16, and 32 bits in parallel and must follow the handshake protocol. The maximum number of bits accepted or provided during a bus transfer is defined as the port width.

The MPC823 contains an address bus that specifies the transfer's address and a data bus that transfers the data. Control signals indicate the beginning and type of the cycle, as well as the address space and size of the transfer. The selected device then controls the length of the cycle with the signal used to terminate the cycle. A strobe signal for the address bus indicates the validity of the address and provides timing information for the data. The MPC823 bus is synchronous, but the bus and control input signals must be timed to setup and hold times relative to the rising edge of the clock. In this situation, bus cycles can be completed in two clock cycles.

Furthermore, for all inputs, the MPC823 latches the level of the input during a sample window around the rising edge of the clock signal. This window is illustrated in Figure 13-1, where **tsu** and **tho** are the input setup and hold times, respectively. To ensure that an input signal is recognized on a specific falling edge of the clock, the input must be stable during the sample window. If an input makes a transition during the window time period, the level recognized by the MPC823 is not predictable. However, the MPC823 always resolves the latched level to either a logic high or low before using it. In addition to meeting input setup and hold times for deterministic operation, all input signals must obey the protocols described in this section.

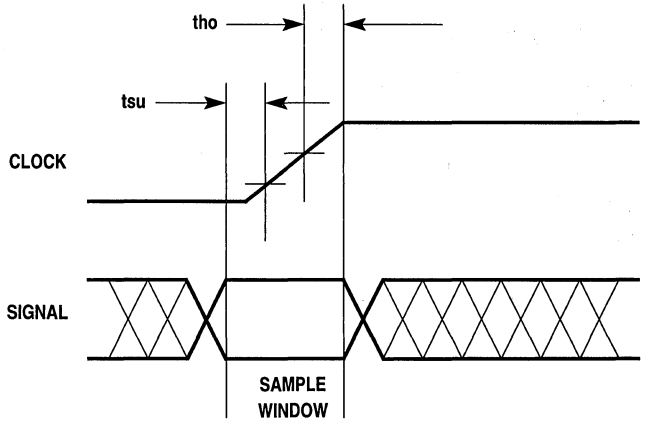


Figure 13-1. Input Sample Window

13.2.1 Control Signals

The MPC823 initiates a bus cycle by driving the address, size, address type, cycle type, and read/write outputs. At the beginning of a bus cycle, the TSIZ0 and TSIZ1 signals are driven with the AT signals. TSIZ0 and TSIZ1 indicate the number of bytes to be transferred during an operand cycle that consists of one or more bus cycles. These signals are valid at the rising edge of the clock in which the $\overline{TS}$ signal is asserted. The RD/$\overline{WR}$ signal determines the direction of the transfer during a bus cycle. Driven at the beginning of a bus cycle, RD/$\overline{WR}$ is valid at the rising edge of the clock in which the $\overline{TS}$ signal is asserted. However, RD/$\overline{WR}$ only transitions when a write cycle is preceded by a read cycle or vice versa. The signal may remain low for consecutive write cycles.

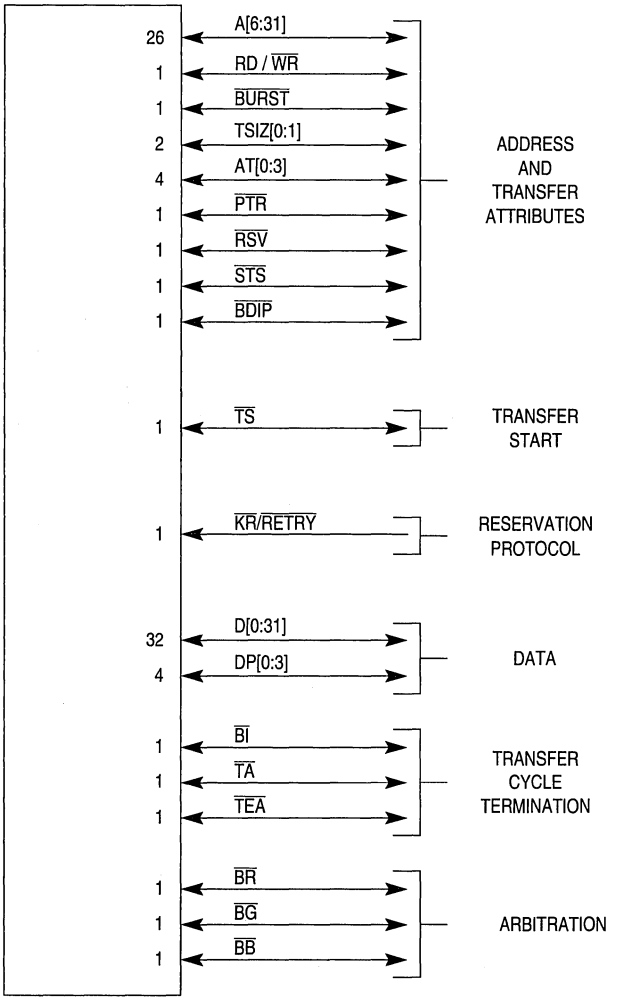


Figure 13-2. MPC823 Bus Signals

13.3 BUS SIGNAL DESCRIPTIONS

The following table decribes each bus interface signal. More detailed descriptions can be found in subsequent sections of this manual.

Table 13-1. Bus Interface Signals

MNEMONIC	PINS	ACTIVE	I/O	DESCRIPTION
ADDRESS AND TRANSFER ATTRIBUTES				
A[6:31]	26	High	O	**Address Bus**—Driven by the MPC823 when it owns the external bus. It specifies the physical address of the bus transaction. These signals can change during a transaction when controlled by the memory controller.
			I	Sampled by the MPC823 when an external device initiates a transaction and the memory controller was configured to handle external master accesses.
RD/$\overline{WR}$	1	High	O	**Read/Write**—Driven by the MPC823 along with the address when it owns the external bus. Driven high indicates that a read access is in progress and driven low indicates that a write access is in progress.
			I	Sampled by the MPC823 when an external device initiates a transaction and the memory controller was configured to handle external master accesses.
$\overline{BURST}$	1	Low	O	**Burst Transfer**—Driven by the MPC823 along with the address when it "owns" the external bus. Driven low indicates that a burst transfer is in progress and driven high indicates that the current transfer is not a burst.
			I	Sampled by the MPC823 when an external device initiates a transaction and the memory controller was configured to handle external master accesses.
TSIZ[0:1]	2	High	O	**Transfer Size**—Driven by the MPC823 along with the address when it owns the external bus. It specifies the data transfer size for the transaction.
			I	Sampled by the MPC823 when an external device initiates a transaction and the memory controller was configured to handle external master accesses.
AT[0:3]	3	High	O	**Address Type**—Driven by the MPC823 along with the address when it owns the external bus. It provides additional information about the address on the current transaction.
			I	Used only for testing purposes.
$\overline{RSV}$	1	Low	O	**Reservation Transfer**—Driven by the MPC823 along with the address when it owns the external bus. It provides additional information about the address on the current transaction.
			I	Used only for testing purposes.
$\overline{PTR}$	1	Low	O	**Program Trace**—Driven by the MPC823 along with the address when it owns the external bus. It provides additional information about the address on the current transaction.
			I	Used only for testing purposes.

Table 13-1. Bus Interface Signals (Continued)

MNEMONIC	PINS	ACTIVE	I/O	DESCRIPTION
$\overline{\text{BDIP}}$	1	Low	O	**Burst Data In Progress**—Driven by the MPC823 when it owns the external bus. It is part of the burst protocol. Asserted indicates that the second beat in front of the current one is requested by the master. This signal is negated prior to the end of a burst to terminate the burst data phase early.
			I	Used only for testing purposes.
TRANSFER START				
$\overline{\text{TS}}$	1	Low	O	**Transfer Start**—Driven by the MPC823 when it owns the external bus.It indicates the start of a transaction on the external bus.
			I	Sampled by the MPC823 when an external device initiates a transaction and the memory controller was configured to handle external master accesses.
$\overline{\text{STS}}$	1	Low	O	**Special Transfer Start**—Driven by the MPC823 when it owns the external bus. It indicates the start of a transaction on the external bus or an internal transaction in show cycle mode.
RESERVATION PROTOCOL				
$\overline{\text{KR}}/\overline{\text{RETRY}}$	1	Low	I	**Kill Reservation/Retry**—When a bus cycle is initiated by a **stwcx** instruction that was issued by the core to a nonlocal bus on which the storage reservation has been lost, this signal is used by the nonlocal bus interface to back-off the cycle. Refer to **Section 13.4.10 Storage Reservation Protocol**. For a regular transaction, this signal is driven by the slave device to indicate that the MPC823 has to relinquish ownership of the bus and retry the cycle.
DATA				
D[0:31]	32	High	I/O	**Data Bus**—The data bus has the following byte lane assignments: Data Byte — Byte Lane D(0:7) — 0 D(8:15) — 1 D(16:23) — 2 D(24:31) — 3
			O	Driven by the MPC823 when it owns the external bus and has initiated a write transaction to a slave device. For single beat transactions, if external A(6:31) and TSIZ(0:1) do not select the byte lanes for transfer, they will not supply valid data.
			I	Driven by the slave in a read transaction. For single beat transactions, if external A(6:31) and TSIZ(0:1) do not select the byte lanes for transfer, they will not be sampled by the MPC823. It is also sampled by the MPC823 when the external master acquires the bus.

Table 13-1. Bus Interface Signals (Continued)

MNEMONIC	PINS	ACTIVE	I/O	DESCRIPTION
DP[0:3]	4	High	I/O	**Parity Bus**—Each parity signal corresponds to each one of the data bus lanes: Data Bus Byte — Parity Line D(0:7) — DP0 D(8:15) — DP1 D(16:23) — DP2 D(24:31) — DP3
			O	Driven by the MPC823 when it owns the external bus and has initiated a write transaction to a slave device. Each parity signal has the parity value (even or odd) of the corresponding data bus byte. For single beat transactions, if external A(6:31) and TSIZ(0:1) do not select the byte lanes for transfer, they will not have a valid parity line.
			I	Driven by the slave in a read transaction. Each parity signal is sampled by the MPC823 and checked (if enabled) against the expected value parity value (even or odd) of the corresponding data bus byte. For single beat transactions, if external A(6:31) and TSIZ(0:1) do not select the byte lanes for transfer, they will not be sampled by the MPC823 and its parity signals will not be checked.
TRANSFER CYCLE TERMINATION				
$\overline{TA}$	1	Low	I	**Transfer Acknowledge**—Driven by the slave device the current transaction was addressed to. It indicates that the slave has received the data on the write cycle or returned the data on the read cycle. If the transaction is a burst, $\overline{TA}$ should be asserted for each one of the transaction beats.
			O	Driven by the MPC823 when the slave device is controlled by the on-chip memory controller.
$\overline{TEA}$	1	Low	I	**Transfer Error Acknowledge**—Driven by the slave device the current transaction was addressed to. It indicates that an error condition has occurred during the bus cycle.
			O	Driven by the MPC823 when the internal bus monitor detects an erroneous bus condition.
$\overline{BI}$	1	Low	I	**Burst Inhibit**—Driven by the slave device the current transaction was addressed to. It indicates that the current slave does not support burst mode.
			O	Driven by the MPC823 when the slave device is controlled by the on-chip memory controller.

Table 13-1. Bus Interface Signals (Continued)

MNEMONIC	PINS	ACTIVE	I/O	DESCRIPTION
ARBITRATION				
$\overline{BR}$	1	Low	I	**Bus Request**—When the internal arbiter is asserted, it indicates that an external master is requesting the bus.
			O	Driven by the MPC823 when the internal arbiter is disabled and the chip is not parked.
$\overline{BG}$	1	Low	O	**Bus Grant**—When the internal arbiter is enabled, the MPC823 asserts this signal to indicate that an external master can assume ownership of the bus and begin a bus transaction. The $\overline{BG}$ signal should be qualified by the master requesting the bus to ensure it is the bus owner: Qualified $\overline{BG}$ = $\overline{BG}$ & ~ $\overline{BB}$
			I	When the internal arbiter is disabled, the $\overline{BG}$ is sampled and properly qualified by the MPC823 when an external bus transaction is to be executed by the chip.
$\overline{BB}$	1	Low	O	**Bus Busy**—When the internal arbiter is enabled, the MPC823 asserts this signal to indicate that it is the current owner of the bus. When the internal arbiter is disabled, it will assert this signal after the external arbiter grants the chip ownership of the bus and it is ready to start the transaction.
			I	When the internal arbiter is enabled, the MPC823 samples this signal to get an indication of when the external master ended its bus tenure ($\overline{BB}$ negated). When the internal arbiter is disabled, the $\overline{BB}$ is sampled to properly qualify the $\overline{BG}$ line when an external bus transaction is to be executed by the chip.

NOTE: O indicates an output from the MPC823 and I indicates an input.

13.4 BUS INTERFACE OPERATION

The MPC823 generates a system clock output (CLKOUT) that sets the frequency of operation for the bus interface. Internally, the MPC823 uses a phase-lock loop (PLL) circuit to generate a master clock for all of the core circuitry, which is phase-locked to the CLKOUT output signal.

All signals for the MPC823 bus interface are specified with respect to the rising-edge of the external CLKOUT and are guaranteed to be sampled as inputs or changed as outputs with respect to that edge. Since the same clock edge is referenced for driving or sampling the bus signals, the possibility of clock skew could exist between various modules in a system because of routing or using multiple clock lines. It is your responsibility to handle any clock skew problems that could occur as a result of layout, lead-length, and physical routing.

13.4.1 Basic Transfers

The basic transfer protocol defines the sequence of actions that must occur on the MPC823 bus to perform a complete bus transaction. The chronological sequence or phase of a typical bus transfer is as follows:

1. Arbitration
2. Address transfer
3. Data transfer
4. Termination

This protocol provides for an arbitration phase and an address and data transfer phase. The arbitration phase specifies the master that initiates the next transaction. The address phase specifies the address for the transaction and the transfer attributes that describe the transaction. The data phase performs the transfer of data. It can transfer a single beat of data (4 bytes or less) for nonburst operations, a 4-beat burst of data, an 8-beat burst of data, or a 16-beat burst of data.

13.4.2 Single Beat Transfers

During the data transfer phase, data is transferred from master to slave on write cycles or from slave to master on read cycles. On a write cycle, the master drives the data as soon as it can, but never before the cycle following the address transfer phase. The master has to take into consideration the "one dead clock cycle" when switching between drivers to avoid electrical contention. The master can stop driving the data bus as soon as it samples the $\overline{TA}$ line asserted on the rising edge of the CLKOUT. On a read cycle the master accepts the data bus contents as valid at the rising edge of the CLKOUT in which the $\overline{TA}$ signal is sampled asserted.

13.4.2.1 SINGLE BEAT READ FLOW. The basic read cycle begins with a bus arbitration, followed by the address transfer and the data transfer. The handshakes are illustrated in the following diagrams as applicable to the fixed transaction protocol.

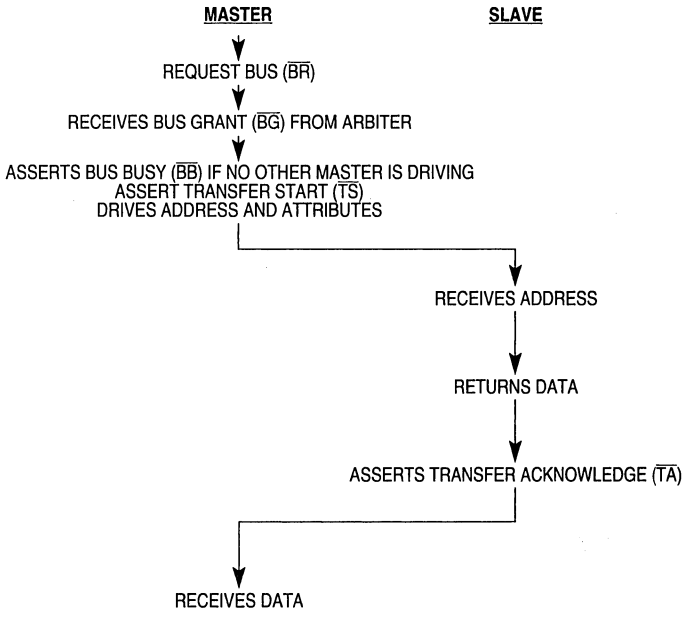


Figure 13-3. Basic Flow Diagram of a Single Beat Read Cycle

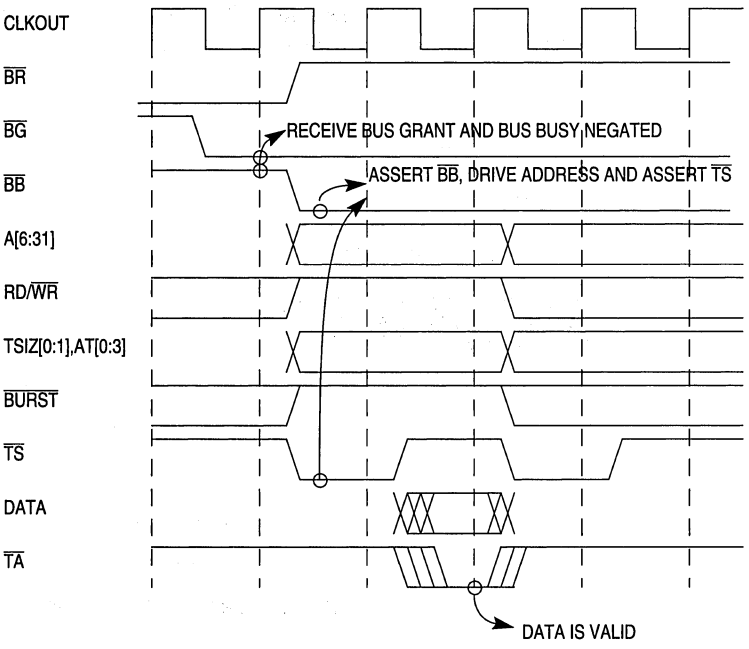


Figure 13-4. Single Beat Read Cycle–Basic Timing–Zero Wait States

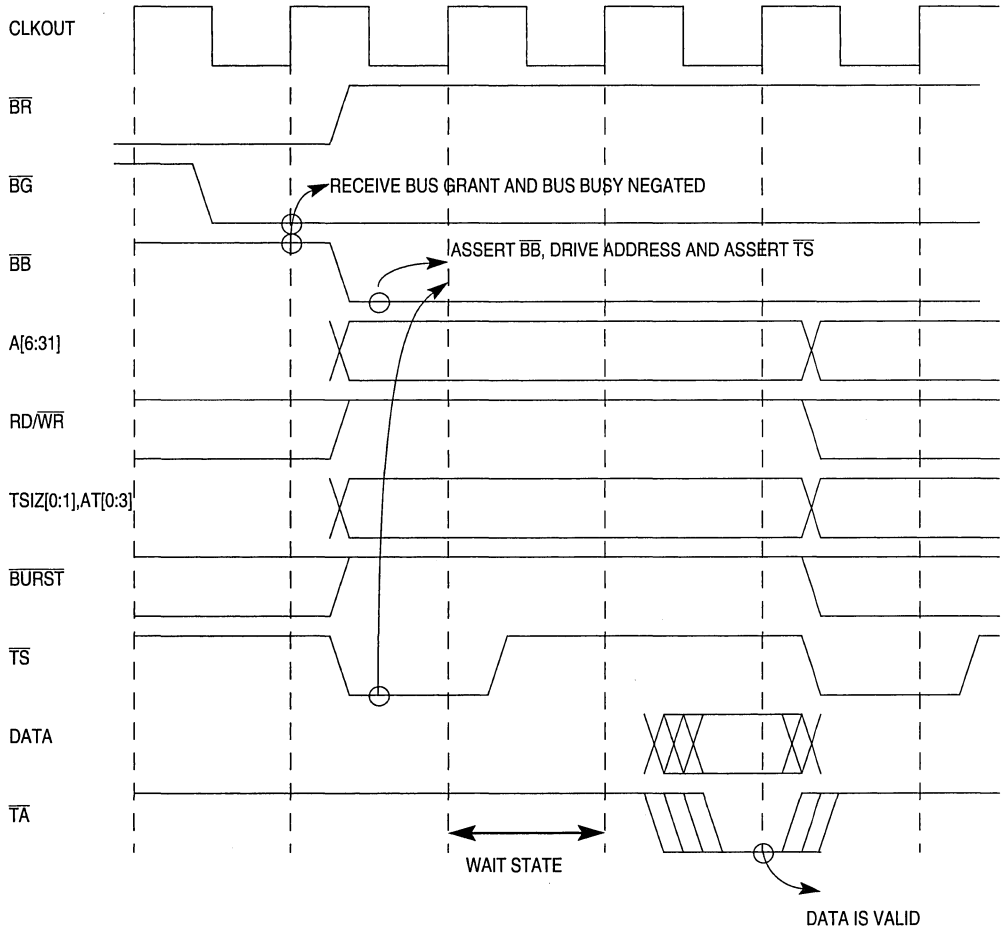


Figure 13-5. Single Beat Read Cycle–Basic Timing–One Wait State

13.4.2.2 SINGLE BEAT WRITE FLOW. The basic write cycle begins with a bus arbitration, followed by the address transfer and the data transfer. The handshakes are illustrated in Figure 13-6, Figure 13-7, Figure 13-8, and Figure 13-9 as applicable to the fixed transaction protocol.

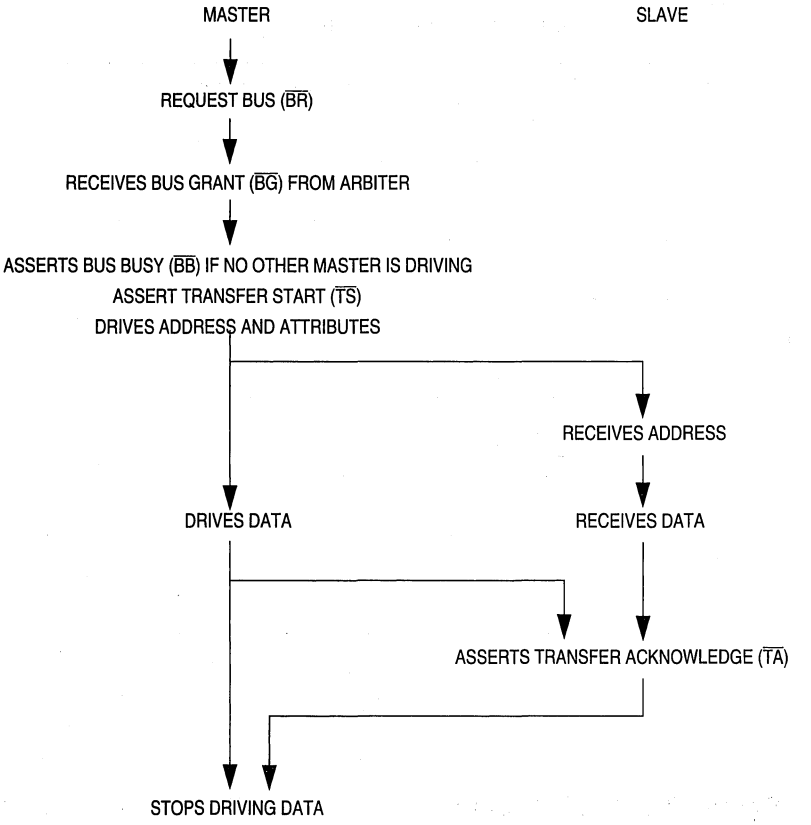


Figure 13-6. Basic Flow Diagram of a Single Beat Write Cycle

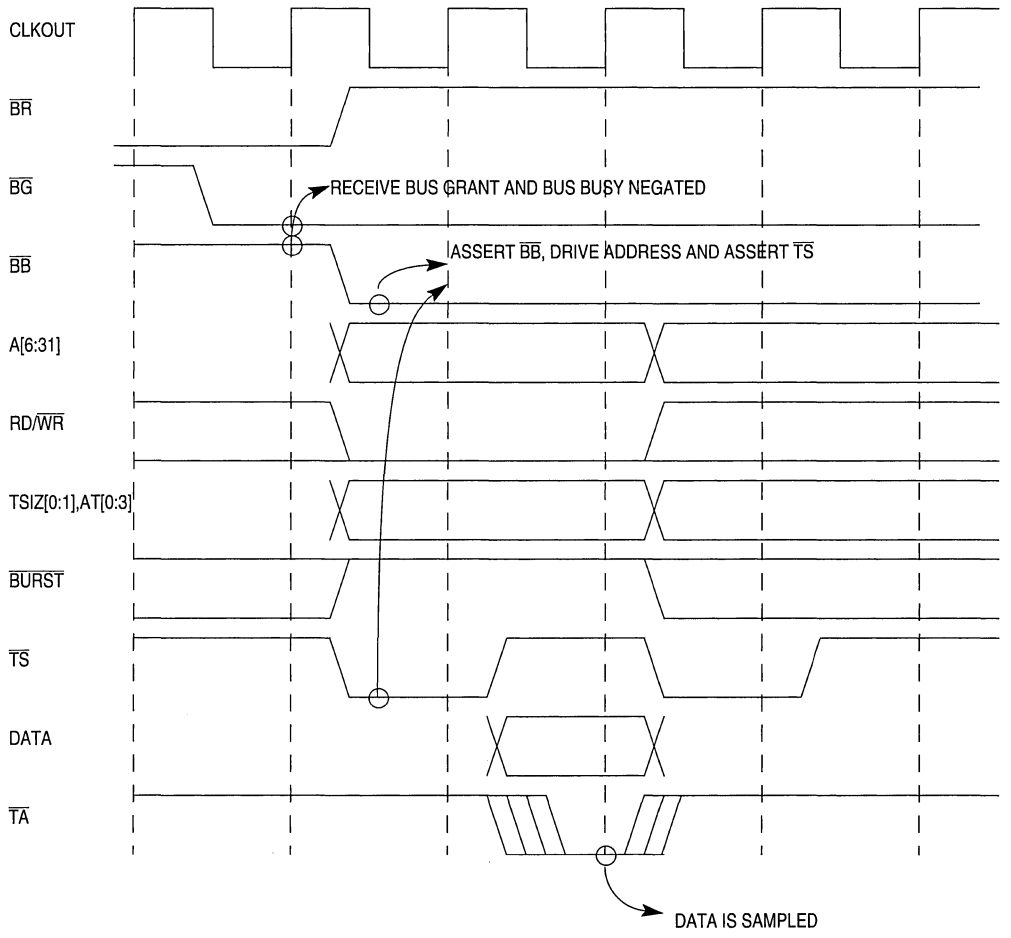


Figure 13-7. Single Beat Write Cycle–Basic Timing–Zero Wait States

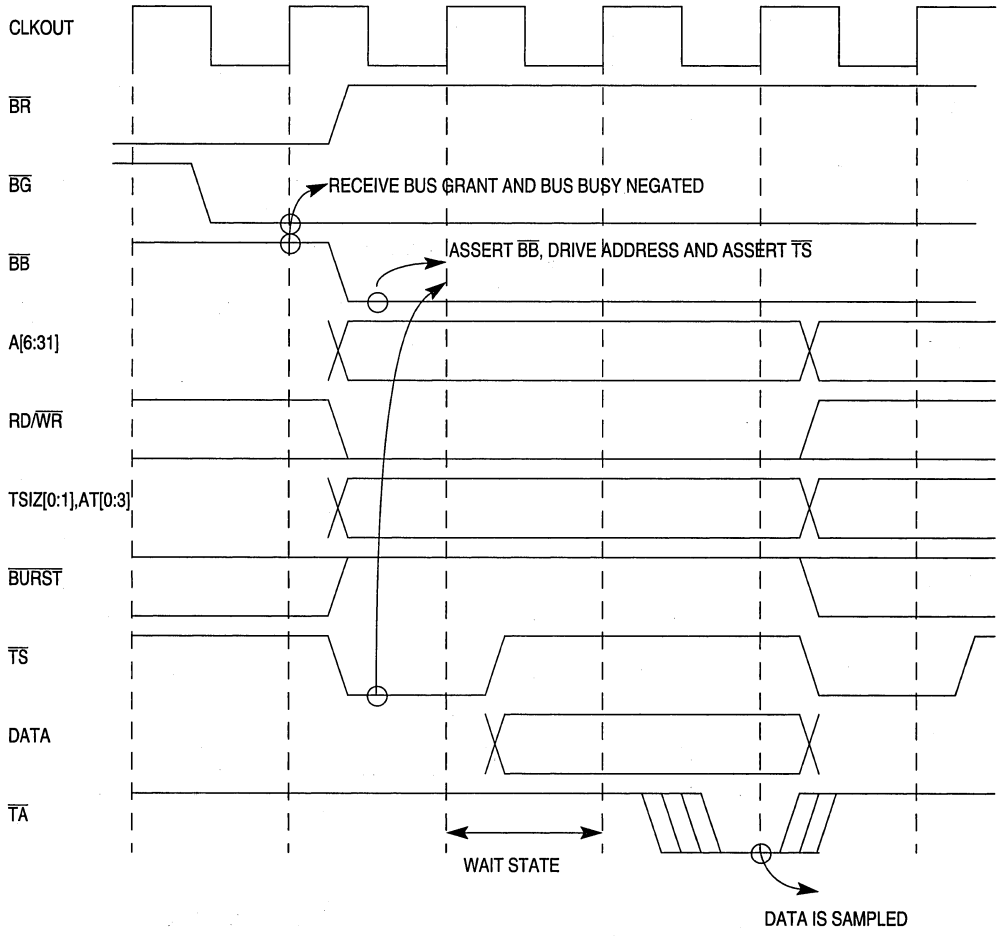


Figure 13-8. Single Beat Write Cycle of One Wait State

A typical single beat transfer assumes that the external memory has a 32-bit port size. The MPC823 provides an effective mechanism for interfacing with 16-bit port size memories and 8-bit port size memories, thus allowing transfers to these devices when they are controlled by the internal memory controller. The port size (PS) timing shown in the following figures is representative of the PS field in **Section 15 Memory Controller**.

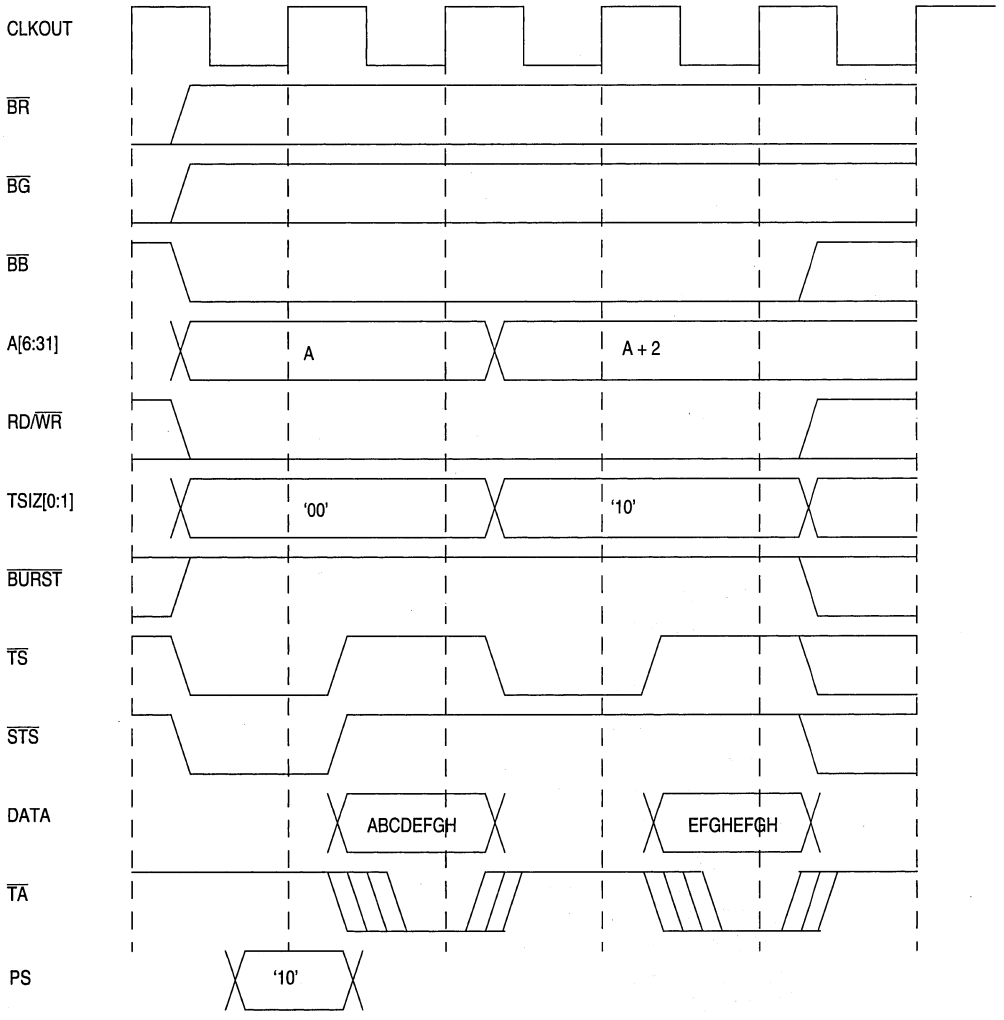


Figure 13-9. Single Beat, 32-Bit Data, Write Cycle From a 16-Bit Port Size

13.4.3 Burst Transfers

The MPC823 uses burst transfers to access 16-byte operands. A burst accesses a block of 16 bytes that must be aligned to a 16-byte memory boundary by supplying a starting address that points to the critical words and requiring the memory device to sequentially drive/sample each word on the data bus. The selected slave device must internally increment the external A[28:29] signal (or A[30] for a 16-bit port size slave device) of the supplied address for each transfer, thus causing the address to wrap around at the end of the 4-word block. The address and transfer attributes supplied by the MPC823 remain stable during the transfers and the selected device terminates each transfer by driving/sampling the word on the data bus and asserting the $\overline{TA}$ signal.

The MPC823 also supports burst-inhibited transfers for slave devices that are unable to support bursting. For this type of bus cycle, the selected slave device supplies/samples the first word the MPC823 points to and asserts the $\overline{BI}$ signal with $\overline{TA}$ for the first transfer of the burst access. The MPC823 responds by terminating the burst and accessing the remainder of the 16-byte block, thus using three read/write cycle bus (each one for a word) for a 32-bit port width slave, seven read/write cycle bus for a 16-bit port width slave, or fifteen read/write cycle bus for an 8-bit port width slave.

Burst transfers assume that the external memory has a 32-bit port size. The MPC823 provides an effective mechanism for interfacing with 16- and 8-bit port size memories that allow burst transfers to these devices when they are controlled by the internal memory controller. The MPC823 attempts to initiate a burst transfer as normal. If the slave device responds to a cycle prior to the $\overline{TA}$ signal for the first beat, its port size is 8 or 16 bits and the MPC823 completes a burst of 8- or 16-bit beats. Effectively, each of the data beats of the burst transfers only 1 or 2 bytes. This 8- or 16-beat burst is also considered an atomic transaction, so the MPC823 will not allow other unrelated master accesses or bus arbitration to intervene between the transfers.

13.4.4 The Burst Mechanism

The MPC823 burst mechanism consists of one signal indicating that the cycle is a burst cycle, one indicating the duration of the burst data, and another signal indicating whether the slave is burstable. These signals are in addition to the basic signals of the bus. At the start of the burst transfer, the master drives the address, address attributes, and $\overline{BURST}$ signal to indicate that a burst transfer is being initiated, along with the assertion of the $\overline{TS}$ signal. If the slave is burstable, it negates the $\overline{BI}$ signal. If the slave cannot burst, it must assert the $\overline{BI}$ signal. During the data phase of a burst write cycle the master drives the data. It also asserts the $\overline{BDIP}$ signal if it intends to drive a subsequent data beat after the current data beat. When the slave has received the data, it asserts the $\overline{TA}$ signal to let the master know it is ready for the next data transfer. The master again drives the next data and asserts or negates the $\overline{BDIP}$ signal. If the master does not intend to drive another data beat after the current one, it negates the $\overline{BDIP}$ signal to let the slave know that the next subsequent data beat transfer is the last data of the burst write transfer. During the data phase of a burst read cycle, the master receives data from the addressed slave. If the master needs more than one data, it asserts the $\overline{BDIP}$ signal. When the data is received prior to the last data, the master negates the $\overline{BDIP}$ signal. Thus, the slave stops driving new data after it receives the negation of the $\overline{BDIP}$ signal at the rising edge of the clock. See Figure 13-10 for details.

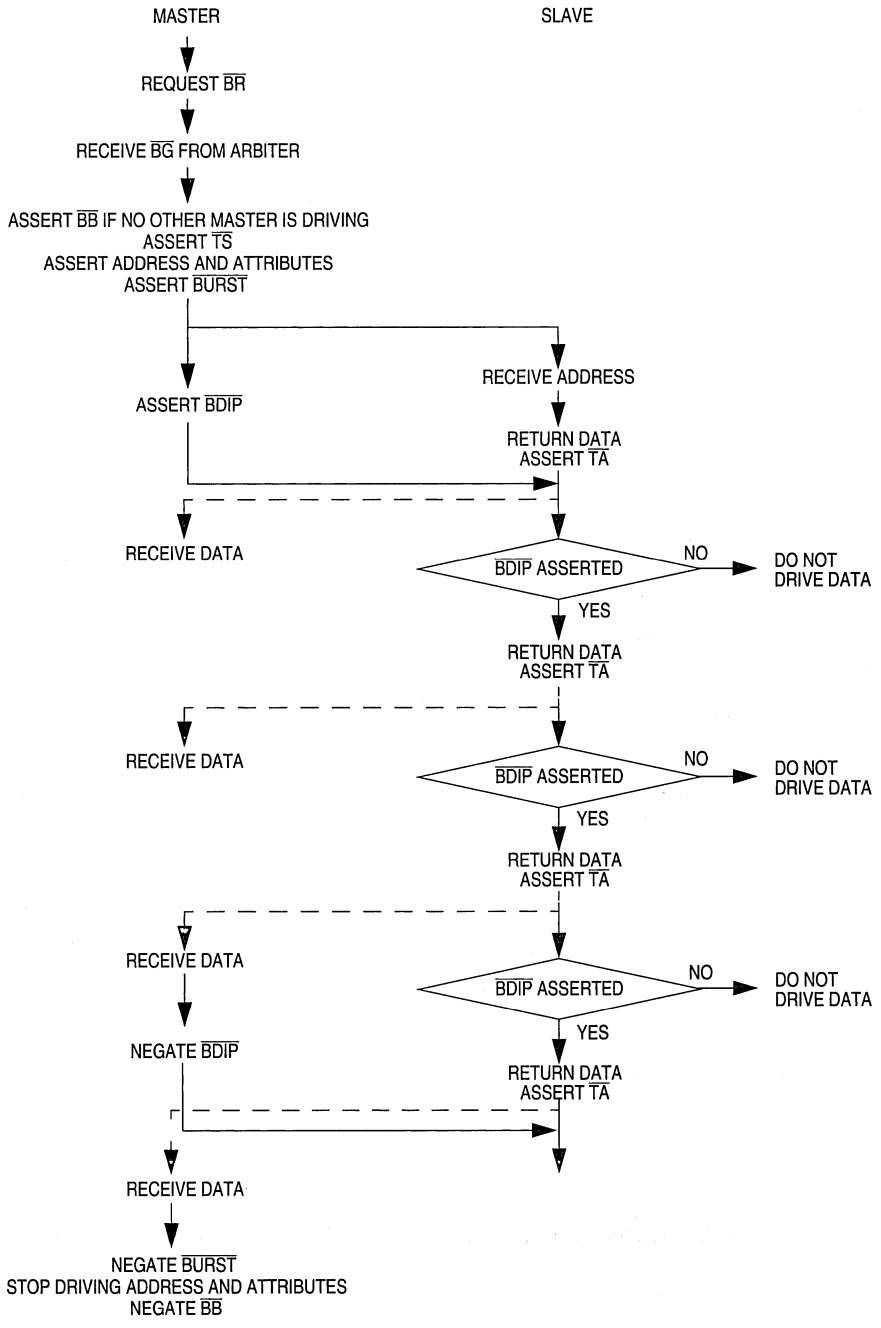


Figure 13-10. Basic Flow Diagram Of A Burst Read Cycle

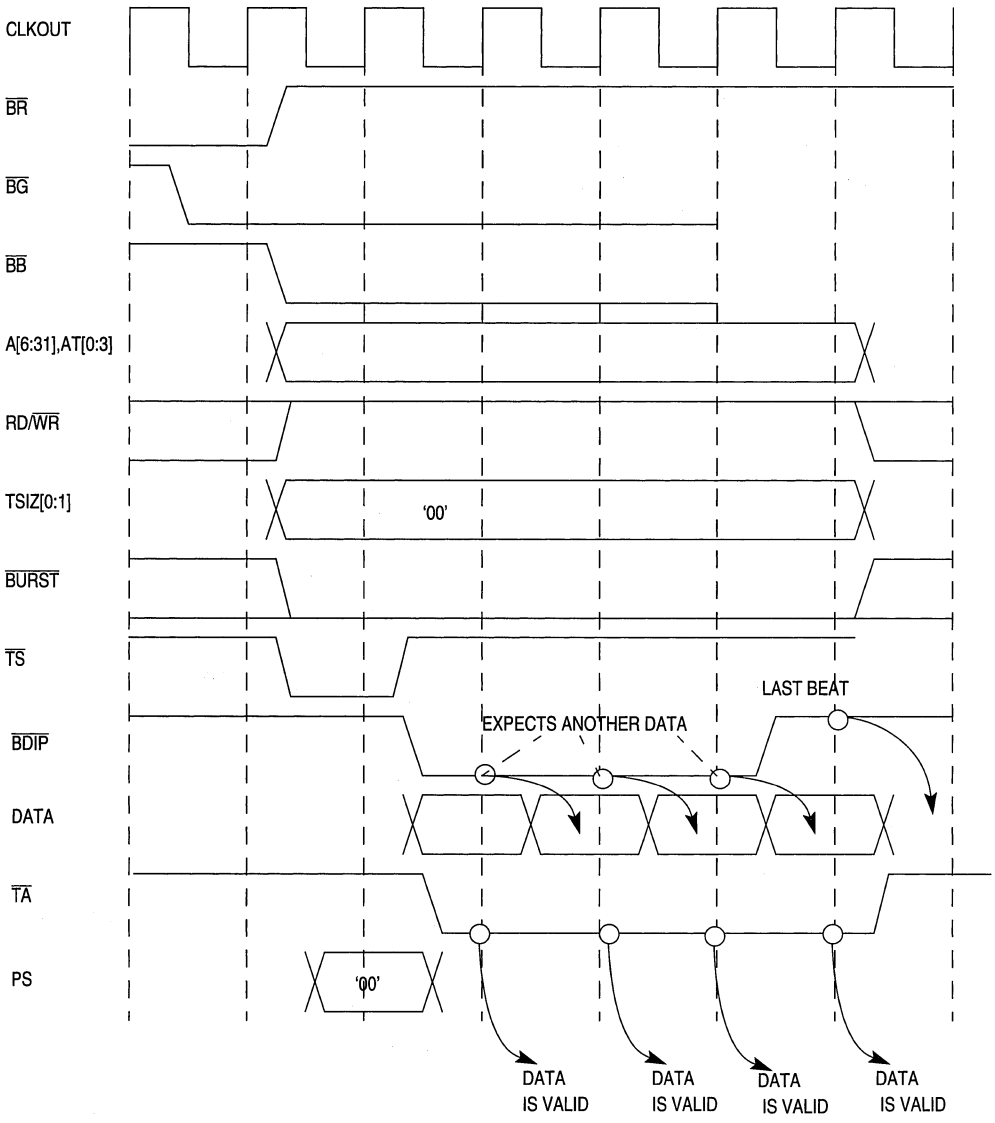


Figure 13-11. Burst-Read Cycle–32-Bit Port Size–Zero Wait State

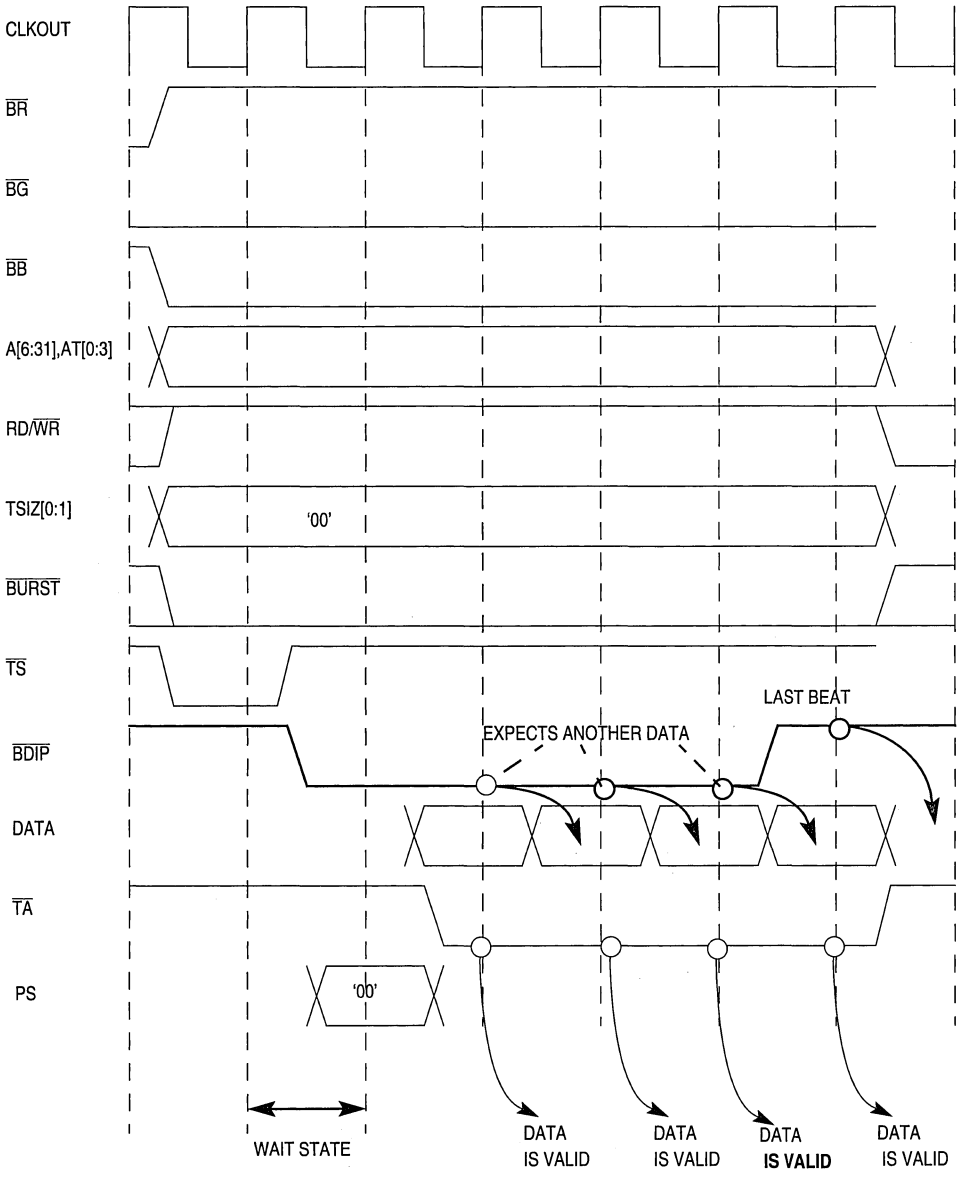


Figure 13-12. Burst-Read Cycle–32-Bit Port Size–One Wait State

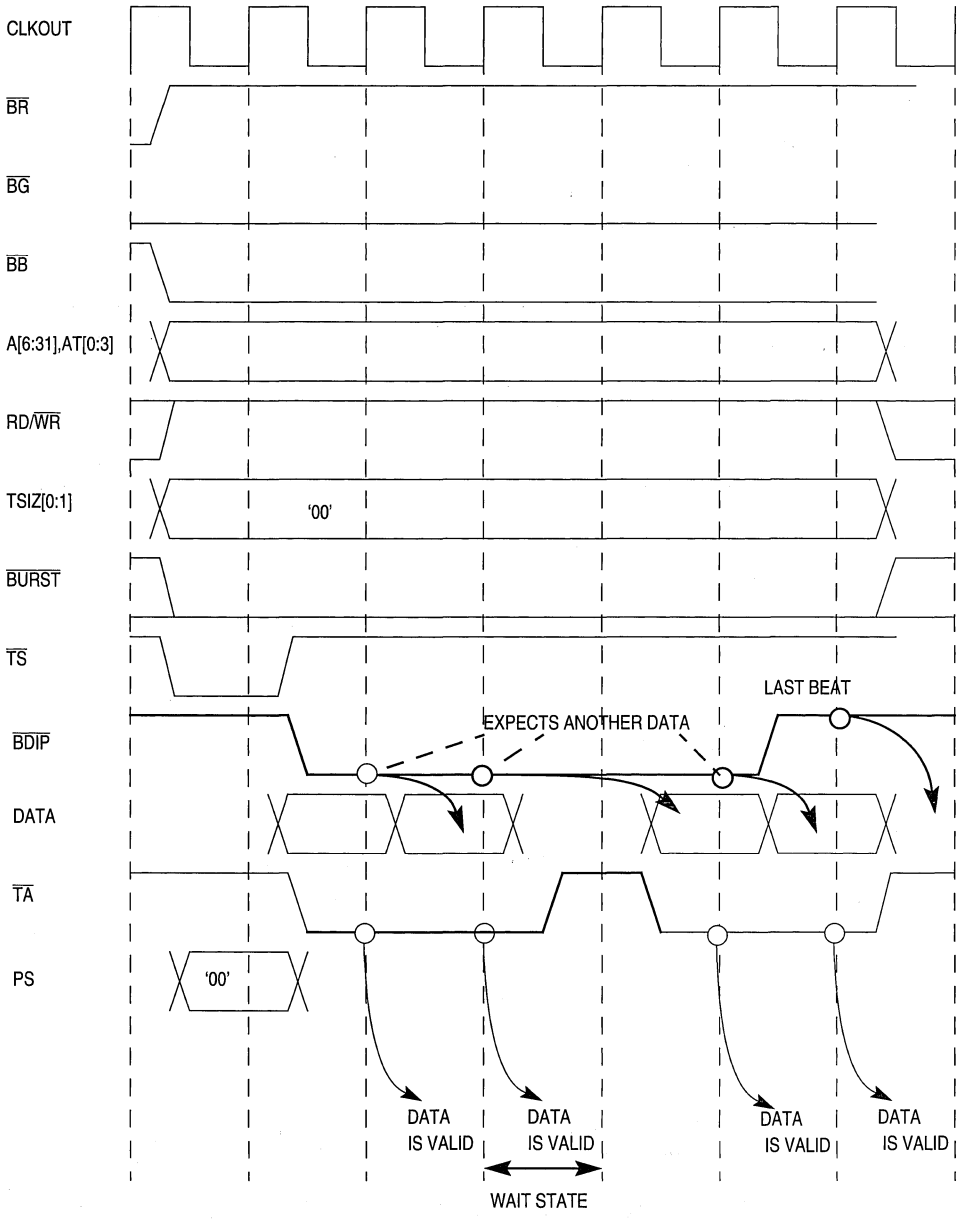


Figure 13-13. Burst-Read Cycle–32-Bit Port Size–Wait States Between Beats

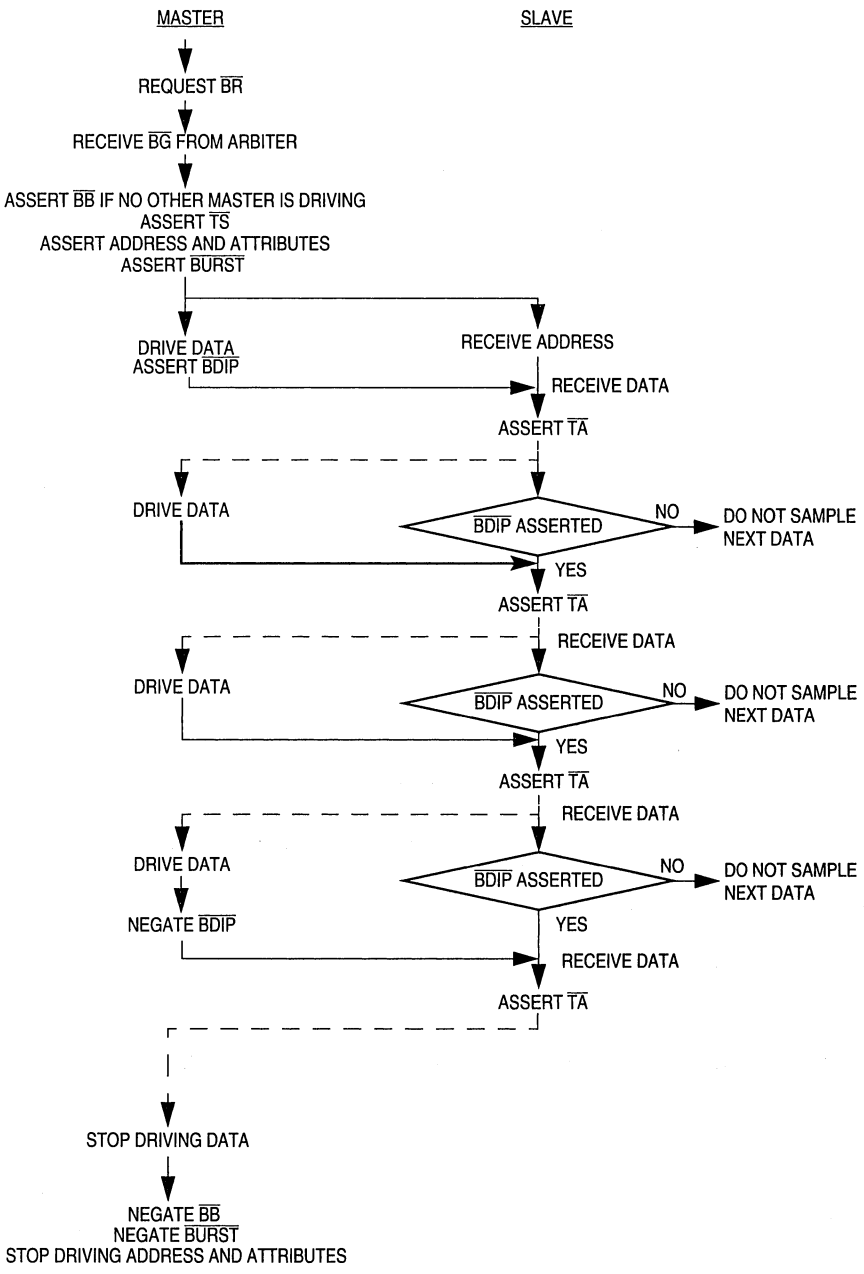


Figure 13-14. Basic Flow Diagram of a Burst Write Cycle

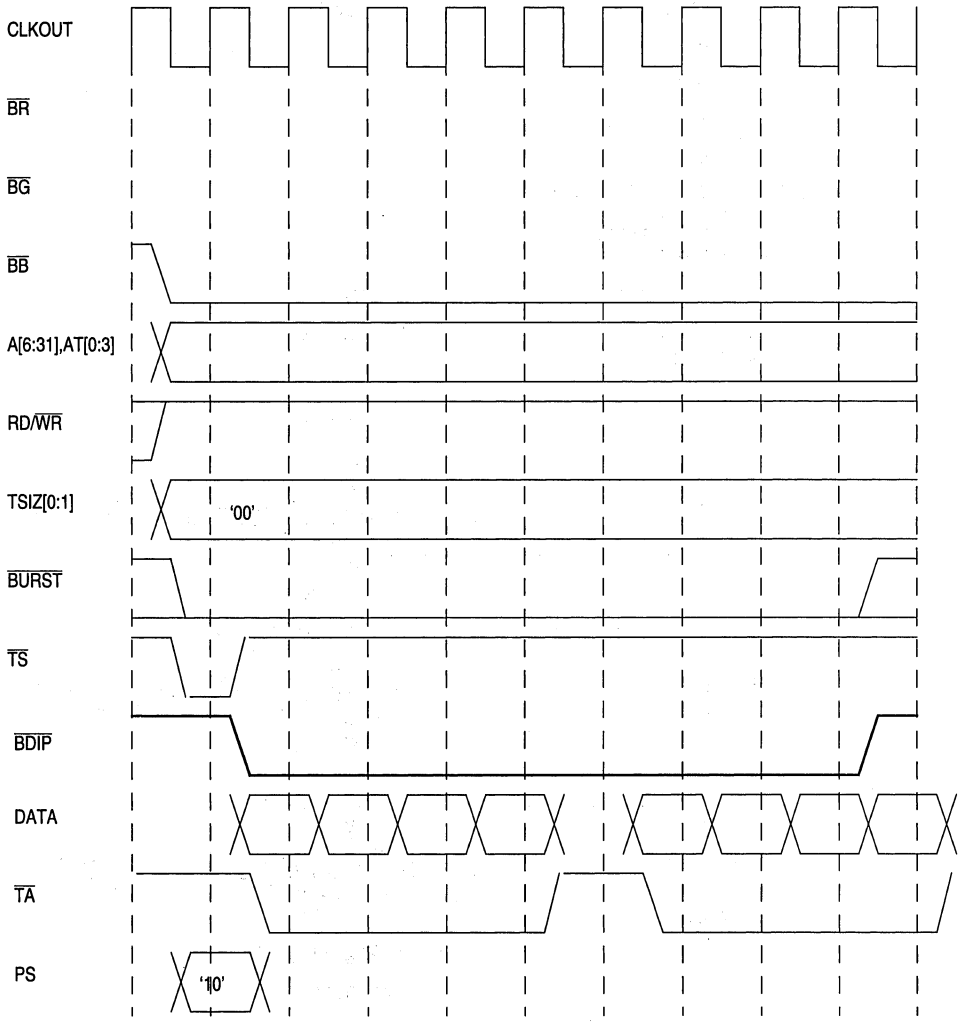


Figure 13-15. Burst-Read Cycle–16-Bit Port Size–One Wait State Between Beats

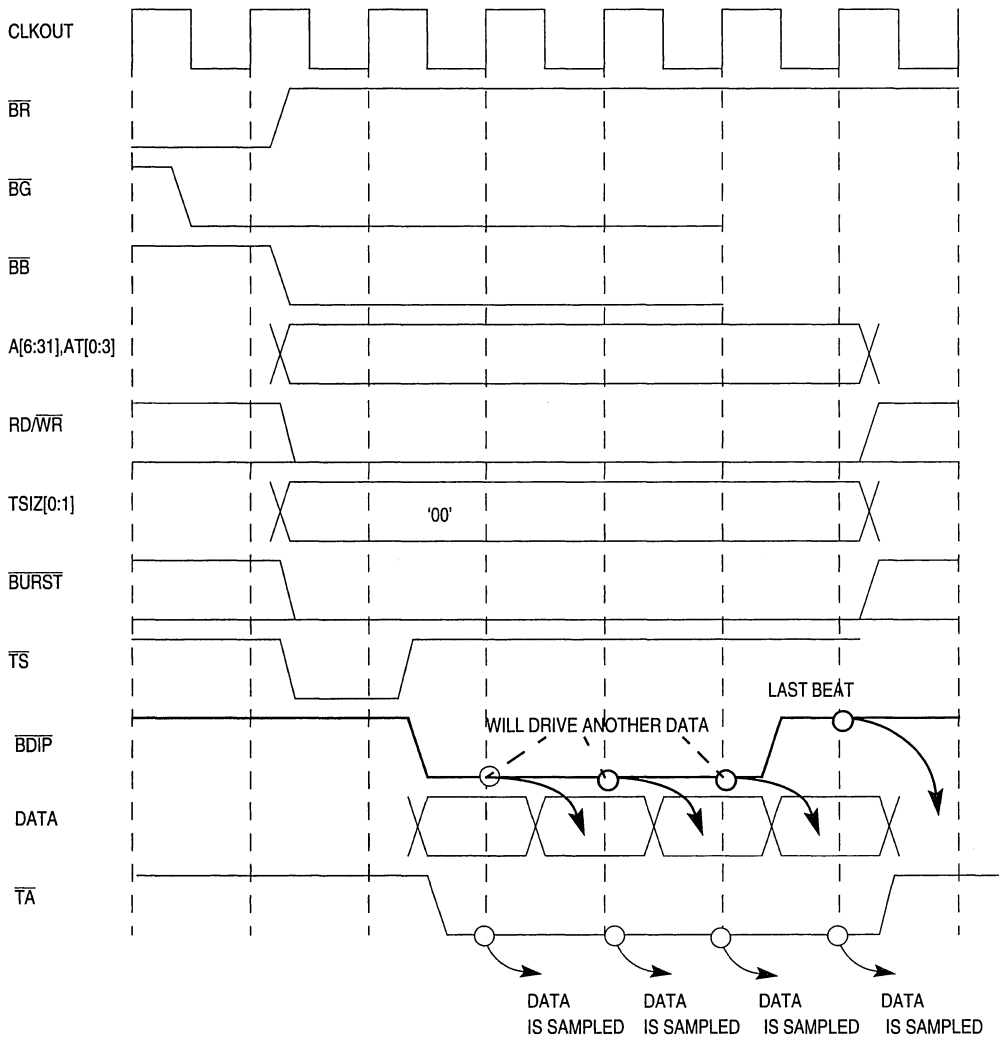


Figure 13-16. Burst-Write Cycle–32-Bit Port Size–Zero Wait States

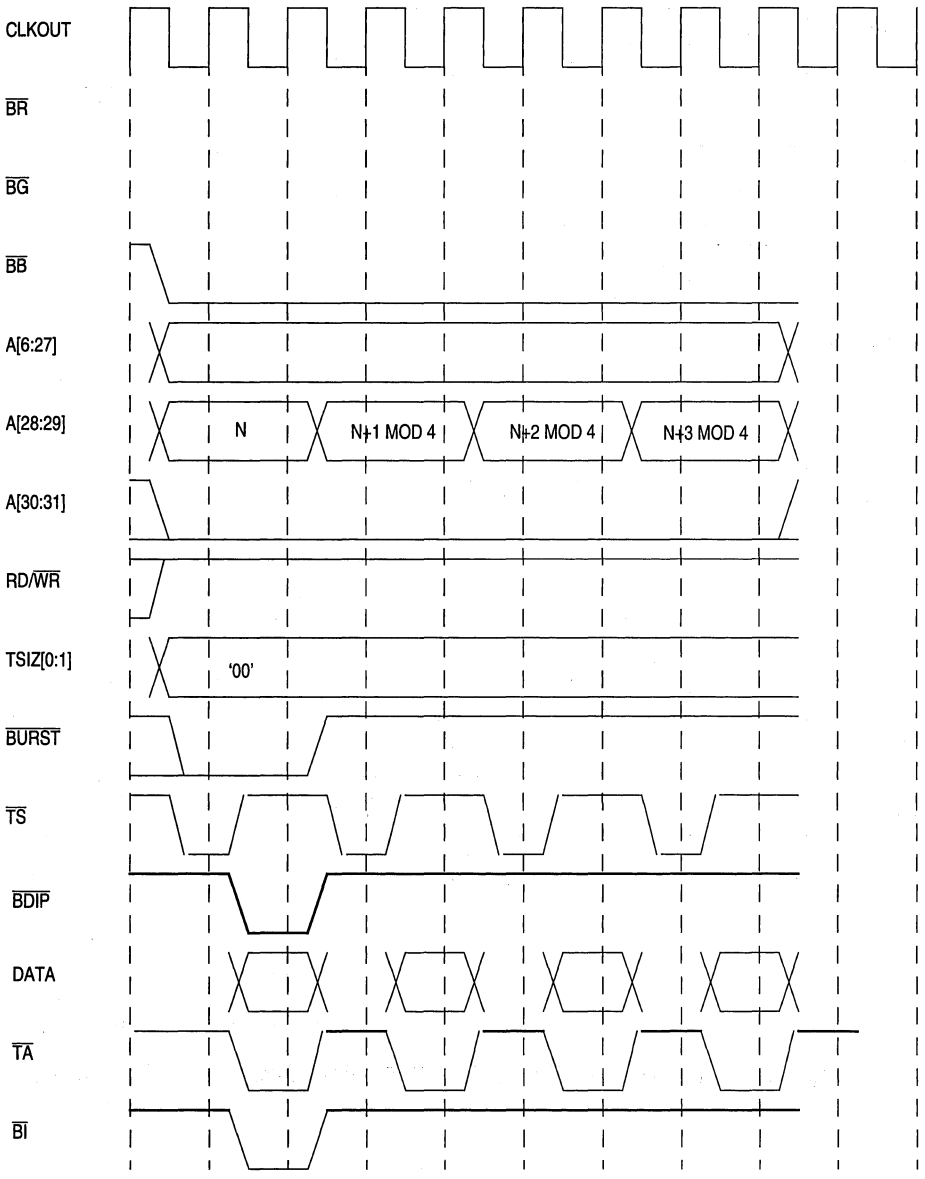


Figure 13-17. Burst-Inhibit Cycle–32-Bit Port Size

13.4.5 Transfer Alignment and Packaging

The MPC823 external bus only supports natural address alignment that forces the following restrictions:

- Byte access can have any address alignment
- Half-word access must have address bit 31 equal to 0
- Word access must have address 30-31 equal to 0
- For burst access must have address 30-31 equal to 0

The MPC823 can perform operand transfers through its 32-bit data port. If the transfer is controlled by the internal memory controller, the MPC823 can support 8- and 16-bit data port sizes. The bus requires that the portion of the data bus used for a transfer to or from a particular port size to be fixed. A 32-bit port must reside on data bus bits 0-31, a 16-bit port must reside on bits 0-15, and an 8-bit port must reside on bits 0-7. The MPC823 always tries to transfer the maximum amount of data on all bus cycles and for a word operation it always assumes that the port is 32 bits wide at the beginning of the bus cycle. In Figure 13-18 and Figure 13-19 and Table 13-2 and Table 13-3, the following operand conventions have been adopted:

- OP0 is the most-significant byte of a word operand and OP3 is the least-significant byte.
- The two bytes of a half-word operand are OP0 (most-significant) and OP1 or OP2 (least-significant) and OP3, depending on the address of the access.
- The single byte of a byte-length operand is OP0, OP1, OP2, or OP3, depending on the address of the access.

Note: Although this is a 32-bit machine, only 26 of the bits are visible outside the chip.

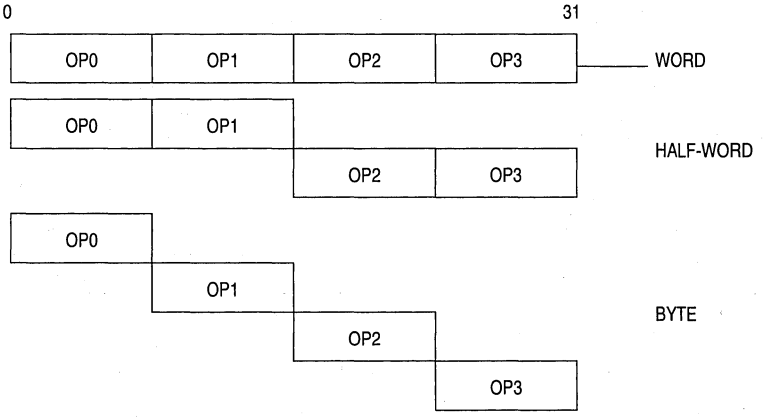


Figure 13-18. Internal Operand Representation

Figure 13-19 illustrates the device connections on the data bus.

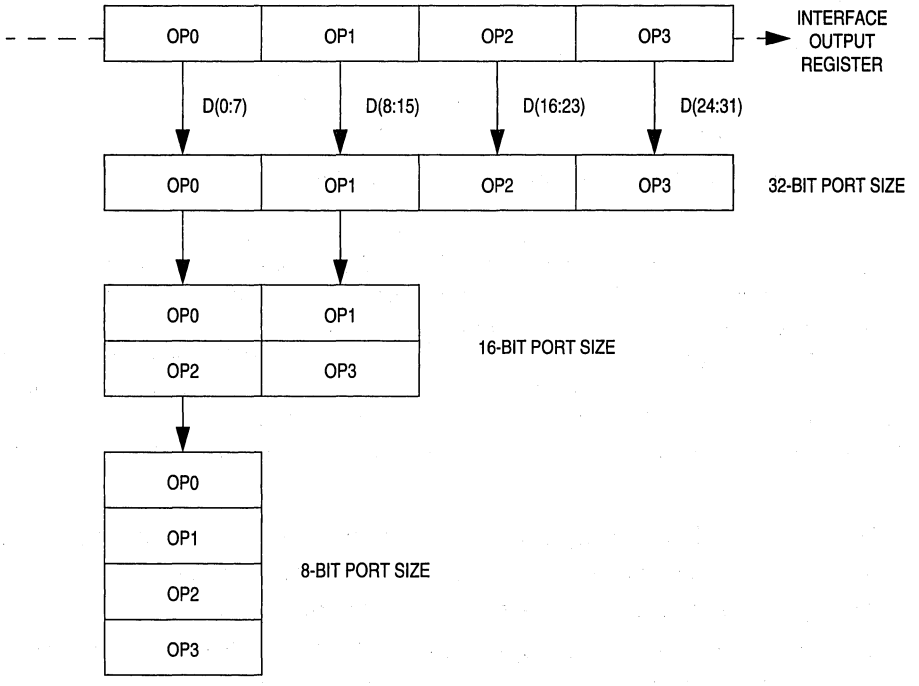


Figure 13-19. Interface To Different Port Size Devices

Table 13-2 lists the bytes for read cycles required on the data bus.

Table 13-2. Data Bus Requirements For Read Cycles

TRANSFER SIZE	TSIZE [0:1]		INTERNAL ADDRESS		32-BIT PORT SIZE				16-BIT PORT SIZE		8-BIT PORT SIZE
			A30	A31	D0–D7	D8–D15	D16–D23	D24–D31	D0–D7	D8–D15	D0–D7
Byte	0	1	0	0	OP0	—	—	—	OP0	—	OP0
	0	1	0	1	—	OP1	—	—	—	OP1	OP1
	0	1	1	0	—	—	OP2	—	OP2	—	OP2
	0	1	1	1	—	—	—	OP3	—	OP3	OP3
Half-Word	1	0	0	0	OP0	OP1	—	—	OP0	OP1	OP0
	1	0	1	0	—	—	OP2	OP3	OP2	OP3	OP2
Word	0	0	0	0	OP0	OP1	OP2	OP3	OP0	OP1	OP0

NOTE: — Denotes that a byte is not required during that read cycle.

Table 13-3 lists the patterns of the data transfer for write cycles when accesses are initiated by the MPC823.

Table 13-3. Data Bus Contents for Write Cycles

TRANSFER SIZE	TSIZE [0:1]		INTERNAL ADDRESS		EXTERNAL DATA BUS PATTERN			
			A30	A31	D0–D7	D8–D15	D16–D23	D24–D31
Byte	0	1	0	0	OP0	—	—	—
	0	1	0	1	OP1	OP1	—	—
	0	1	1	0	OP2	—	OP2	—
	0	1	1	1	OP3	OP3	—	OP3
Half-Word	1	0	0	0	OP0	OP1	—	—
	1	0	1	0	OP2	OP3	OP2	OP3
Word	0	0	0	0	OP0	OP1	OP2	OP3

NOTE: — Denotes that a byte is not required during that read cycle.

13.4.6 Arbitration Phase-Related Signals

The external bus design provides for a single bus master, either the MPC823 or an external device. One or more of the external devices on the bus has the capability of becoming bus master for the external bus. Bus arbitration may be handled either by an external central bus arbiter or by the internal on-chip arbiter. In the latter case, the system is optimized for one external bus master besides the MPC823. The arbitration configuration (external or internal) is set at system reset. See **Section 15.6 External Master Support** for more information.

Each bus master must have $\overline{BR}$, $\overline{BG}$, and $\overline{BB}$ signals. The device that needs the bus asserts the $\overline{BR}$ signal. The device then waits for the arbiter to assert the $\overline{BG}$ signal. In addition, the new master must look at the $\overline{BB}$ signal to ensure that no other master is driving the bus before it can assert $\overline{BB}$ and assume ownership of the bus. If the arbiter has taken the $\overline{BG}$ away from the master and the master wants to execute a new cycle, the master must rearbitrate before a new cycle can be initiated. The MPC823, however, guarantees data coherency for burst accesses to a small port size. This means that the MPC823 will not release the bus until the transactions (atomic) complete.

Figure 13-20 describes the basic protocol for bus arbitration. For more information, see **Section 12.12.1.1 SIU Module Configuration Register**.

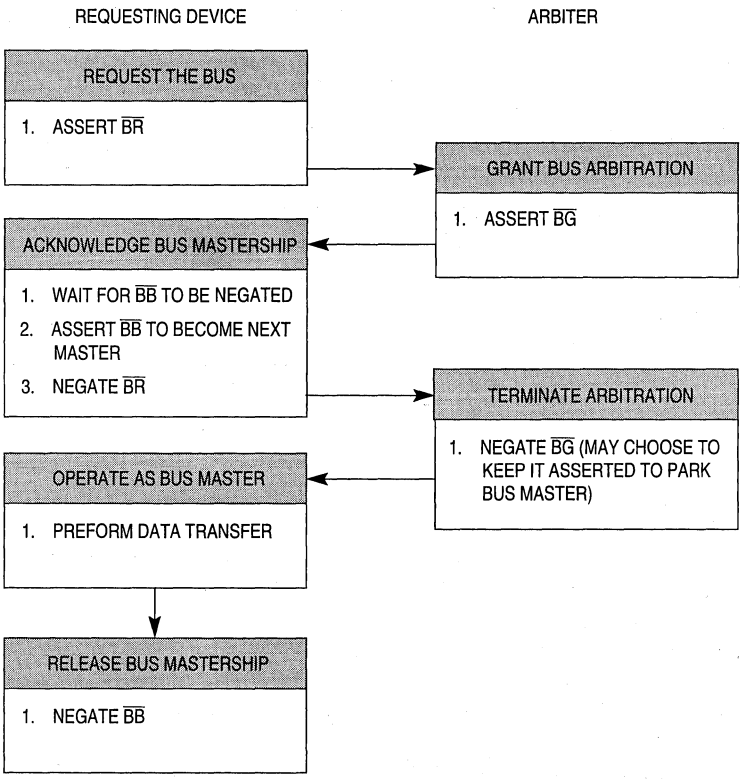


Figure 13-20. Bus Arbitration Flowchart

13.4.6.1 BUS REQUEST SIGNAL. The potential bus master asserts the $\overline{BR}$ signal to request bus mastership. $\overline{BR}$ should be negated once the bus is granted, the bus is not busy, and the new master can drive the bus. If more requests are pending, the master can keep asserting its bus request as long as needed. When configured for external central arbitration, the MPC823 drives this signal when it needs bus mastership. When the internal on-chip arbiter is used, this signal is an input to the internal arbiter and should be driven by the external bus master.

13.4.6.2 BUS GRANT SIGNAL. The $\overline{BG}$ signal is asserted by the arbiter to indicate that the bus is granted to the requesting device. The $\overline{BG}$ signal can be negated after $\overline{BR}$ is negated. The current bus master may choose to keep BG asserted to park the bus and maintain ownership without rearbitrating until another master makes a request. This reduces arbitration time, which then improves performance. When configured for external central arbitration, the $\overline{BG}$ becomes an input signal to the MPC823 from the external arbiter. When the internal on-chip arbiter is used, this signal is an output from the internal arbiter to the external bus master.

13.4.6.3 BUS BUSY SIGNAL. The $\overline{BB}$ signal indicates that the current bus master is using the bus. New masters should not begin transferring until this signal is negated. The bus owner should not relinquish or negate this signal until its transfer is complete. To avoid contention on the $\overline{BB}$ signal, masters should three-state this signal when it gets a logical 1 value. This situation implies that the connection of an external pull-up resistor is needed to ensure that a master acquiring the bus recognizes that the $\overline{BB}$ signal is negated, regardless of how many cycles have passed since the previous master relinquished the bus. Refer to Figure 13-21 for more information.

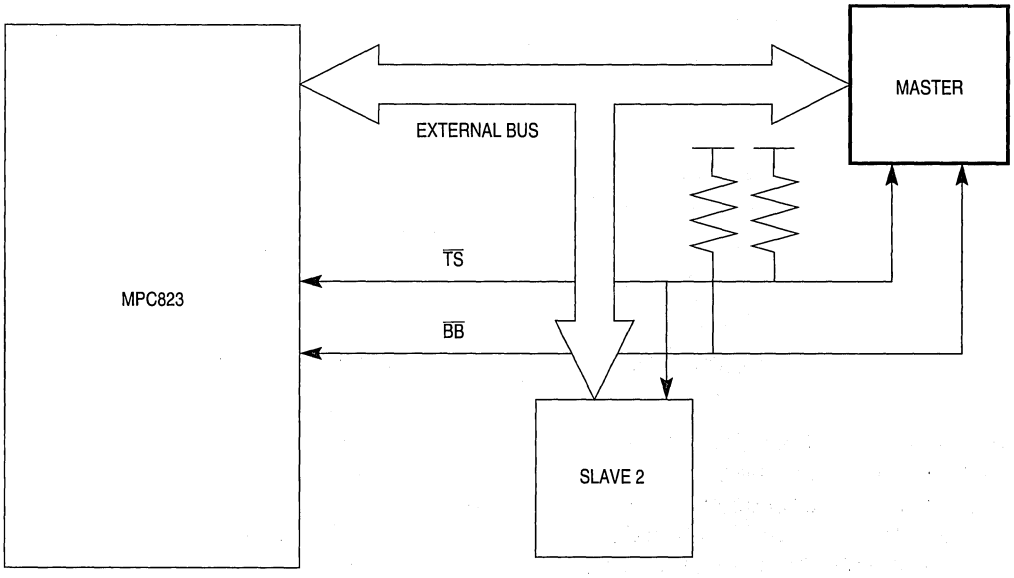


Figure 13-21. Basic Bus Busy Connection

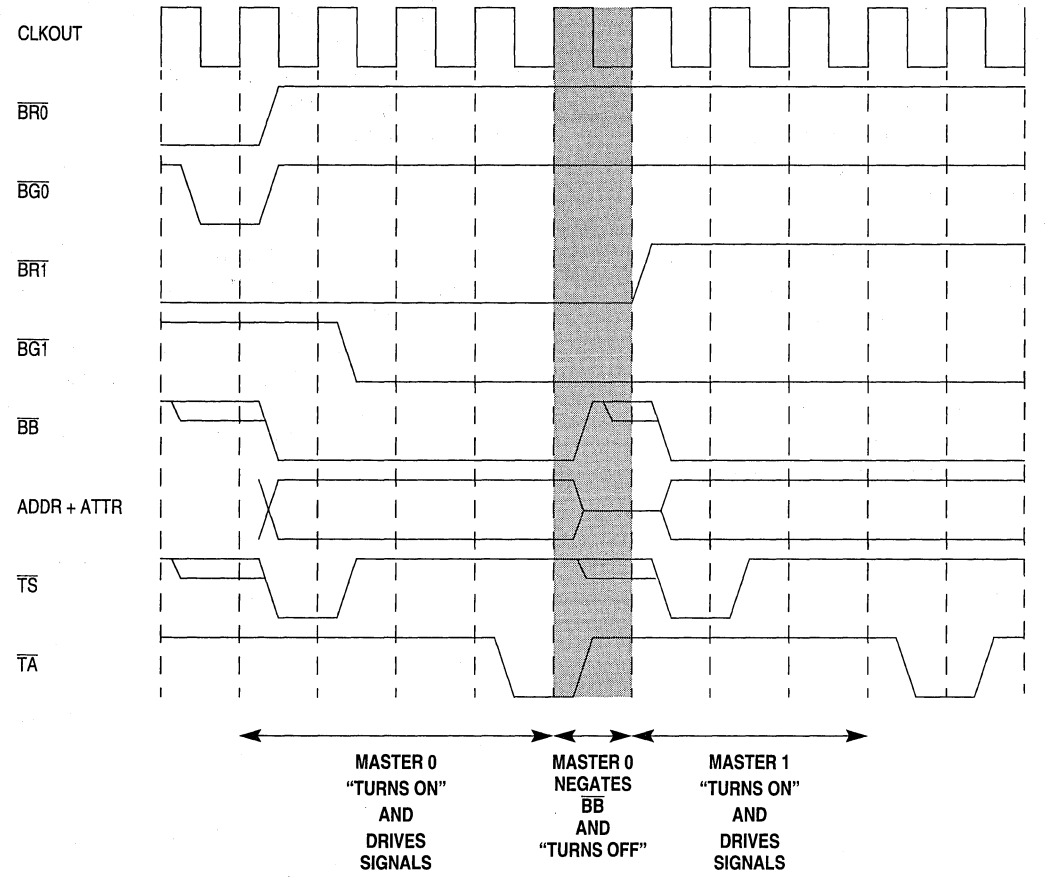


Figure 13-22. Bus Arbitration Timing Diagram

At system reset, the MPC823 can be configured to use the internal bus arbiter and it will be parked on the bus. The priority of the external device relative to the internal MPC823 bus masters is programmed in the SIU module configuration register, as shown in **Section 12.12.1.1 SIU Module Configuration Register**. If the external device requests the bus and the MPC823 does not need it or the external device has priority over the current internal bus master, the MPC823 grants the bus to the external device. Figure 13-23 illustrates the internal finite state machine that implements the arbiter protocol.

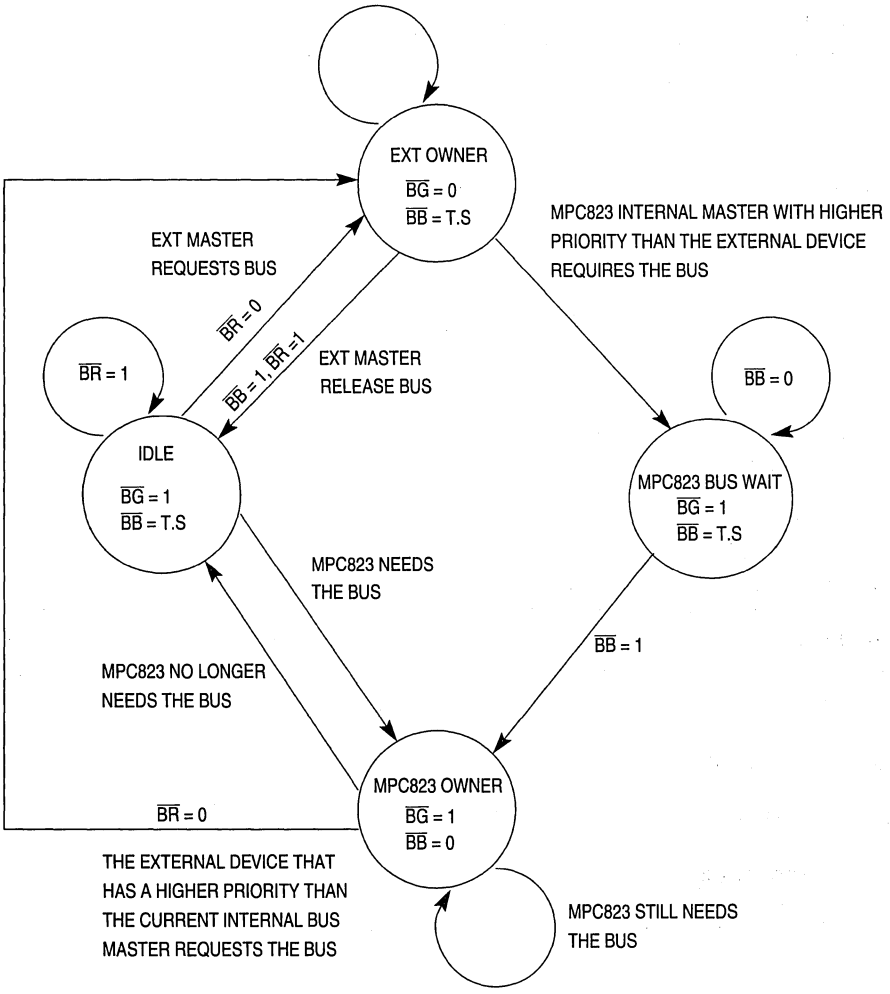


Figure 13-23. Internal Bus Arbitration State Machine

13.4.7 Address Transfer Phase-Related Signals

13.4.7.1 TRANSFER START SIGNAL. The $\overline{TS}$ signal indicates the beginning of a cycle initiated by the bus master. This signal should be asserted by a master only after ownership of the bus is granted by the arbitration protocol. This signal is only asserted for the first clock cycle of the transaction and is negated in the successive clock cycles. The master should three-state this signal when it relinquishes the bus to avoid contention between two or more masters in this signal. This configuration requires an external pull-up resistor to be connected to the $\overline{TS}$ signal. This will prevent a slave from responding to a bogus $\overline{TS}$ assertion. Refer back to Figure 13-21 for more information.

13.4.7.2 ADDRESS BUS. The 26-bit address bus consists of address bits 0–25 and Bit 6 is most-significant. The bus is byte addressable, so each address can address one or more bytes. The address and its attributes are driven on the bus with the $\overline{TS}$ signal and stay valid until the bus master received a $\overline{TA}$ signal from the slave. To distinguish the individual byte, the slave device must observe the TSIZx signals.

Note: Although this is a 32-bit machine, only 26 of the bits are visible outside the chip.

13.4.7.3 TRANSFER ATTRIBUTES. The transfer attributes consist of the RD/$\overline{WR}$, $\overline{BURST}$, TSIZx, ATx, $\overline{STS}$, and $\overline{BDIP}$ signals. These signals, except for the $\overline{BDIP}$, are available at the same time as the address bus.

13.4.7.3.1 Read/Write Signal. When the RD/$\overline{WR}$ signal is high it indicates a read access and when it is low it indicates a write access.

13.4.7.3.2 Burst Signal. The $\overline{BURST}$ signal and the address are driven by the bus master at the beginning of the bus cycle to indicate that the transfer is a burst transfer. The burst size is always 16 bytes. With a 32-bit port size, the burst includes 4 beats. When its port size is 16 bits and controlled by the internal memory controller, the burst includes 8 beats. When its port size is 8 bits and controlled by the internal memory controller, the burst includes 16 beats. The MPC823 bus supports critical data word first for burst. The order of the wraparound goes back to the critical word. For example, assuming data 2 is the critical word:

- Case burst of four beats:

  data 2 ➡ data 3 ➡ data 0 ➡ data 1

- Case burst of eight beats:

  data 2 ➡ data 3 ➡ data 4 ➡ ..... ➡ data 7 ➡ data 0 ➡ data 1

13.4.7.3.3 Transfer Size Signal. The TSIZx signals indicate the size of the requested data transfer and they can be used with the BURST and A[30:31] signals to determine which byte lanes of the data bus are involved in the transfer. For nonburst transfers, the TSIZx signals specify the number of bytes starting from the byte location addressed by the A[30:31] signals. In burst transfers, the value of the TSIZx signal is always 00.

Table 13-4. BURST/TSIZE Encoding

BURST	TSIZx	TRANSFER SIZE
1	01	Byte
1	10	Half-Word
1	11	x
1	00	Word
0	00	Burst (16 bytes)

13.4.7.3.4 Address Space Attributes. The address space attributes consist of the address type (AT[0:3]), PTR, and RSV signals, which are all outputs that indicate one of 16 "address types" to which the address applies. These types are designated as either a normal/alternate master cycle, problem/privilege (user or supervisor), and instruction or data types. The address space signals are valid at the rising edge of the clock in which the STS signal is asserted.

Address space signals reflect the current status of the master originating the access, not necessarily the status in which the original access to this location has occurred. An example of this situation is when a copyback of a dirty line in the data cache occurs after the privilege state of the processor has been changed since the last access to the same line. Functional usage of the ATx, PTR, and RSV signals is for the reservation protocol described in **Section 13.4.10 Storage Reservation Protocol**. Table 13-5 provides the space definition encoded by the STS, TS, ATx, PTR, and RSV signals.

Show cycles are accesses to the core's internal bus devices. These accesses are driven externally for emulation, visibility, and debugging purposes. A show cycle can have one address phase and one data phase (or just an address phase for the instruction show cycles). The cycle can be a write or read access and the data for both the read and write accesses should be driven by the bus master. This is different than the normal bus read and write accesses. The address of the show cycle should be valid on the bus for one clock and the data of the show cycle should be valid on the bus for one clock. The data phase should not require a transfer acknowledge to terminate the bus-show cycle. In a burst show cycle only the first data beat will be shown externally.

13.4.7.3.5 Special Transfer Start Signal. The STS signal is driven by the MPC823 when it owns the external bus. It indicates the start of a transaction on the external bus or an internal transaction in show cycle mode.

Table 13-5. Address Space Definitions

$\overline{STS}$	$\overline{TS}$	AT[0]	AT[1]	AT[2]	AT[3]	$\overline{PTR}$	$\overline{RSV}$	DEFINITIONS
		CORE/CPM	PROBLEM STATE/ PRIVILEGE STATE	INSTRUCTION/ DATA	RESERVATION/ PROGRAM TRACE	PROGRAM TRACE	RESERVATION	
1	x	x	x	x	x	1	1	No Transfer or no first transaction of a transfer
0	x	x	x	x	x	x	x	Start of a transaction
x	0	0	0	0	0	0	1	Core, Normal Instruction, Program Trace, Privilege State
					1	1	1	Core, Normal Instruction, Privilege State
				1	0	1	0	Core, Reservation Data, Privilege State
					1	1	1	Core, Normal Data, Privilege State
			1	0	0	0	1	Core, Normal Instruction, Program Trace, Problem State
					1	1	1	Core, Normal Instruction, Problem State
				1	0	1	0	Core, Reservation Data, Problem State
					1	1	1	Core, Normal Data, Problem State
		1	CH0	CH1	CH2	1	1	No Core, Normal, (CH indicates channel number)

Table 13-5. Address Space Definitions (Continued)

$\overline{STS}$	$\overline{TS}$	AT[0]	AT[1]	AT[2]	AT[3]	$\overline{PTR}$	$\overline{RSV}$	DEFINITIONS
		CORE/CPM	PROBLEM STATE/ PRIVILEGE STATE	INSTRUCTION/ DATA	RESERVATION/ PROGRAM TRACE	PROGRAM TRACE	RESERVATION	
x	1	0	0	0	0	0	1	Core, Show Cycle Address Instruction, Program Trace, Privilege State
					1	1	1	Core, Show Cycle Address Instruction, Privilege State
				1	0	1	0	Core, Reservation Show Cycle Data, Privilege State
					1	1	1	Core, Show Cycle Data, Privilege State
			1	0	0	0	1	Core, Show Cycle Address Instruction, Program Trace, Problem State
					1	1	1	Core, Show Cycle Address Instruction, Problem State
				1	0	1	0	Core, Reservation Show Cycle Data, Problem State
					1	1	1	Core, Show Cycle Data, Problem State
		1	CH0	CH1	CH2	1	1	No Core, Show Cycle Data (CH indicates channel number)

13.4.7.3.6 Burst Data in Progress Signal. The $\overline{\text{BDIP}}$ signal is sent from the master to the slave to indicate that there is a data beat following the current data beat. The master uses this signal to give the slave advanced warning of the remaining data in the burst. By negating the $\overline{\text{BDIP}}$ signal, you can terminate a burst cycle early. Refer to **Section 13.4.2 Single Beat Transfers** and **Section 13.4.4 The Burst Mechanism** for more information.

13.4.8 Data Transfer Phase-Related Signals

13.4.8.1 DATA SIGNAL. The D[0:31] signals are driven by the MPC823 when it owns the external bus and has initiated a write transaction to a slave device. During a read transaction the D[0:31] signals are driven by the slave device. See Table 13-2 for byte lane assignments.

13.4.9 Termination Phase-Related Signals

13.4.9.1 TRANSFER ACKNOWLEDGE SIGNAL. The $\overline{\text{TA}}$ signal indicates the normal completion of a bus transfer. During burst cycles, the slave asserts this signal with every data beat returned or accepted. This signal should be pulled up to V_{DD} with a pull-up resistor.

13.4.9.2 BURST INHIBIT SIGNAL. The $\overline{\text{BI}}$ signal is sent from the slave to the master to indicate that the addressed device does not have burst capability. If this signal is asserted or equal to 0, the master must transfer in multiple cycles and increment the address for the slave to complete the burst transfer. For a system that does not use the burst mode at all, this signal can be permanently tied low. This signal should be pulled up to V_{DD} with a pull-up resistor.

13.4.9.3 TRANSFER ERROR ACKNOWLEDGE SIGNAL. The $\overline{\text{TEA}}$ signal terminates the bus cycle under bus error conditions. The current bus cycle should be aborted. This signal should override any other cycle termination signals, such as the $\overline{\text{TA}}$ signal. This signal should be pulled up to V_{DD} with a pull-up resistor.

13.4.9.4 PROTOCOL FOR TERMINATION SIGNALS. The transfer protocol was defined to avoid electrical contention on signals that can be driven by various sources. To do that, a slave should not drive signals associated with the data transfer until the address phase is completed and it recognizes the address as its own. The slave should disconnect from signals immediately after it has acknowledged the cycle and no later than the termination of the next address phase cycle. This indicates that the termination signals should be connected to power through a pull-up resistor to avoid a situation in which a master samples an undefined value in any of these signals when no real slave is addressed. See Figure 13-24 and Figure 13-25 for more information.

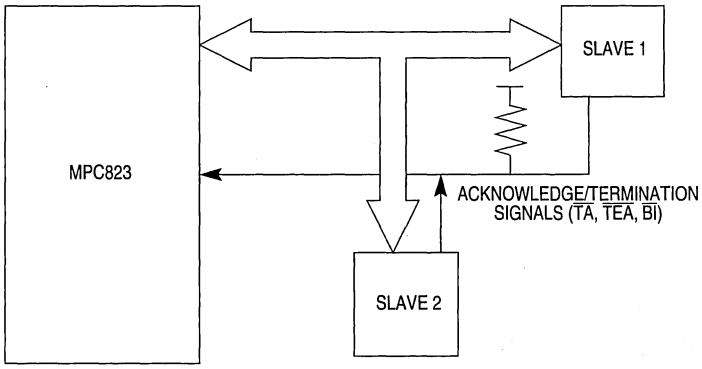


Figure 13-24. Termination Signals Protocol Basic Connection

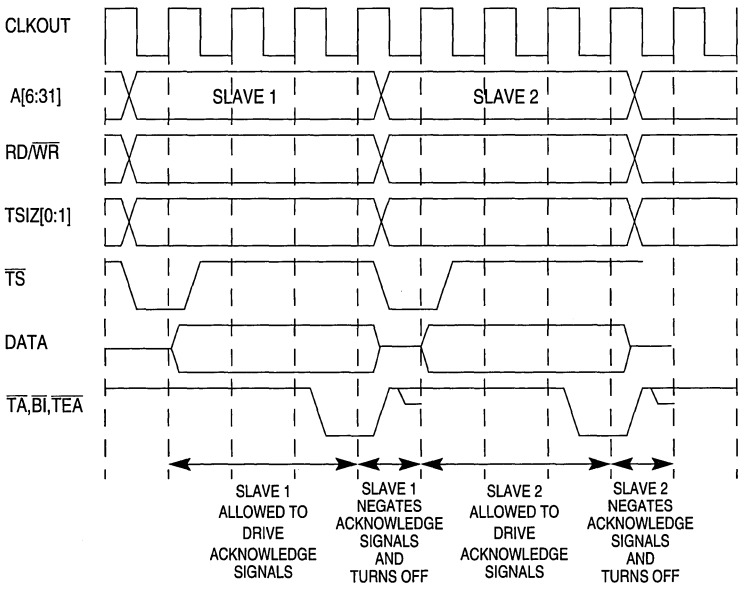


Figure 13-25. Termination Signals Protocol Timing Diagram

13.4.10 Storage Reservation Protocol

The MPC823 storage reservation protocol supports multilevel bus structure. For each local bus, storage reservation is handled by the local reservation logic. The protocol tries to optimize reservation cancellation so that a PowerPC processor is notified of storage reservation loss on a remote bus only when it has issued a **stwcx** cycle to that address. In other words, the reservation loss indication comes as part of the **stwcx** cycle. This method avoids the need to have fast storage reservation loss indication signals routed from every remote bus to every PowerPC master.

The storage reservation protocol makes the following assumptions:

- Each processor has, at most, one reservation "flag"
- **lwarx** sets the reservation "flag"
- **lwarx** by the same processor clears the reservation "flag" related to a previous **lwarx** instruction and again sets the reservation "flag"
- **stwcx** by the same processor clears the reservation "flag"
- A store by the same processor does not clear the reservation "flag"
- Some other processor (or other mechanism) store to the same address as an existing reservation clears the reservation "flag"
- If the storage reservation is lost it is guaranteed that **stwcx** will not modify the storage

The reservation protocol for a single-level (local) bus is illustrated in Figure 13-26. It assumes that external logic on the bus performs the following functions:

- Snoops accesses to all local bus slaves
- Holds one reservation for each local master capable of storage reservations
- Sets the reservation when that master issues a load and reserve request
- Clears the reservation when some other master issues a store to the reservation address

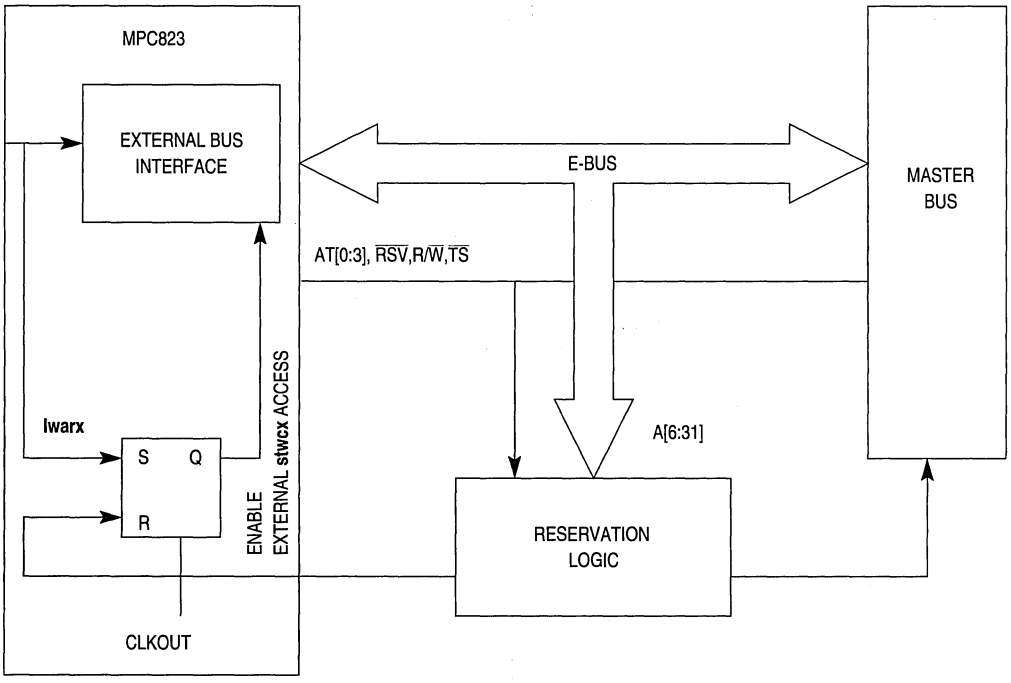


Figure 13-26. Reservation On Local Bus

The local bus interface block implements a reservation "flag" for the local bus master. The reservation "flag" is set by the local bus interface when a load with reservation is issued by the local bus master and the reservation address is located on the remote bus. The "flag" is reset when an alternative master on the remote bus accesses the same location in a write cycle. If the MPC823 begins a memory cycle to the previously reserved address (located in the remote bus) as a result of a **stwcx** instruction, one of the following conditions can occur:

- If the reservation "flag" is set, the local bus interface acknowledges the cycle in a normal way.
- If the reservation "flag" is reset, the local bus interface should assert $\overline{KR}$. However, the local bus interface should either not perform the remote bus write access or abort it if the remote bus supports aborted cycles. The failure of the **stwcx** instruction is reported to the core.

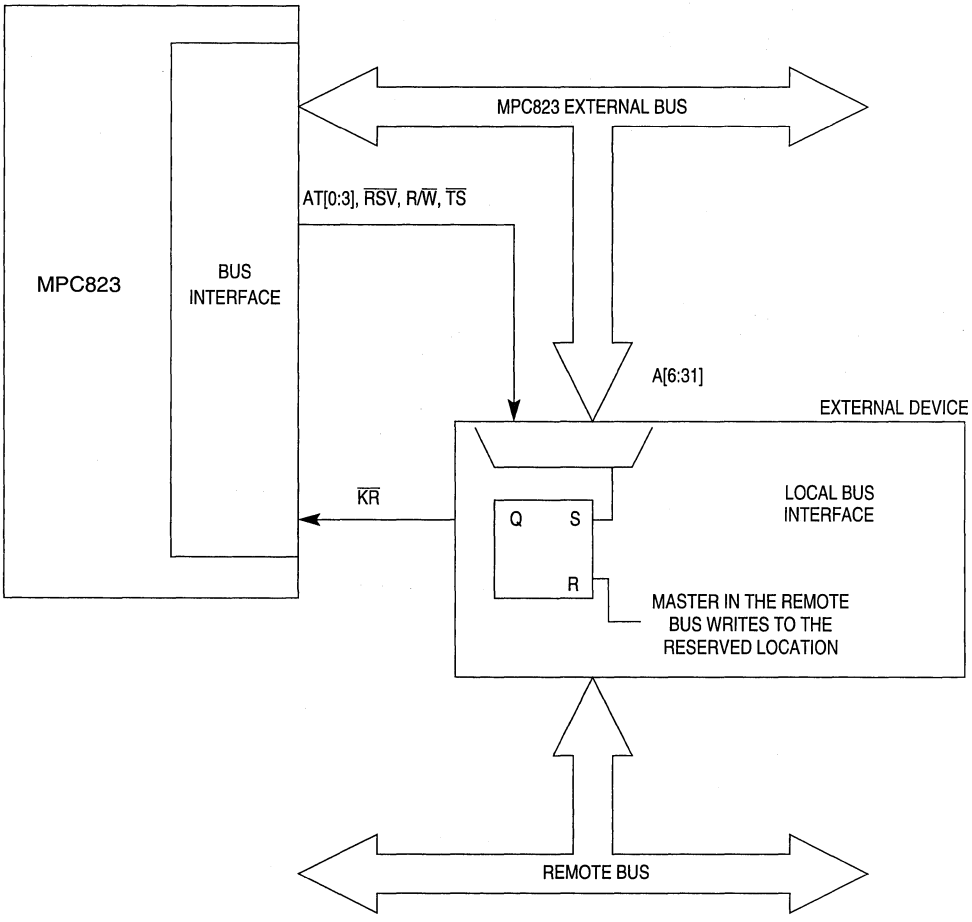


Figure 13-27. Reservation On Multilevel Bus Hierarchy

13.4.11 Exception Control Cycles

The MPC823 bus architecture requires the $\overline{TA}$ signal to be asserted from an external device to indicate that the bus cycle is complete. $\overline{TA}$ is not asserted when one of the following conditions occur:

- The external device does not respond
- Other application-dependent errors occur

The external circuitry can provide $\overline{TEA}$ when no device responds by asserting $\overline{TA}$ within an appropriate period of time after the MPC823 initiates the bus cycle. This allows the cycle to terminate and the processor to enter exception processing for the error condition.

To properly control termination of a bus cycle for a bus error, $\overline{TEA}$ must be asserted simultaneously or before $\overline{TA}$ is asserted. $\overline{TEA}$ should be negated before the second rising edge after it was sample-asserted to avoid detecting an error for the next initiated bus cycle. $\overline{TEA}$ is an open-drain pin that allows the wire-OR of any different error generation sources.

13.4.11.1 RETRY SIGNAL. When an external device asserts the $\overline{RETRY}$ signal during a bus cycle, the MPC823 enters a sequence in which it terminates the current transaction, relinquishes ownership of the bus, and retries the cycle using the same address, address attributes, and data. Figure 13-28 illustrates the behavior of the MPC823 when the $\overline{RETRY}$ signal is detected as a termination of a transfer. The figure illustrates that when the internal arbiter is enabled, the MPC823 negates the $\overline{BB}$ signal and asserts the $\overline{BG}$ signal in the clock cycle following retry detection. This allows any external master to gain bus ownership. In the next clock cycle, a normal arbitration procedure may occur. The figure also shows that the external master did not use the bus, so the MPC823 initiates a new transfer with the same address and attributes as before. In Figure 13-29 the same situation is illustrated to show that the MPC823 is working with an external arbiter. In the clock cycle after the CPU recognizes that the $\overline{RETRY}$ signal is asserted, the $\overline{BR}$ and $\overline{BB}$ signals are negated. One clock cycle later, the normal arbitration procedure may occur. This input signal requires a pull-up resistor.

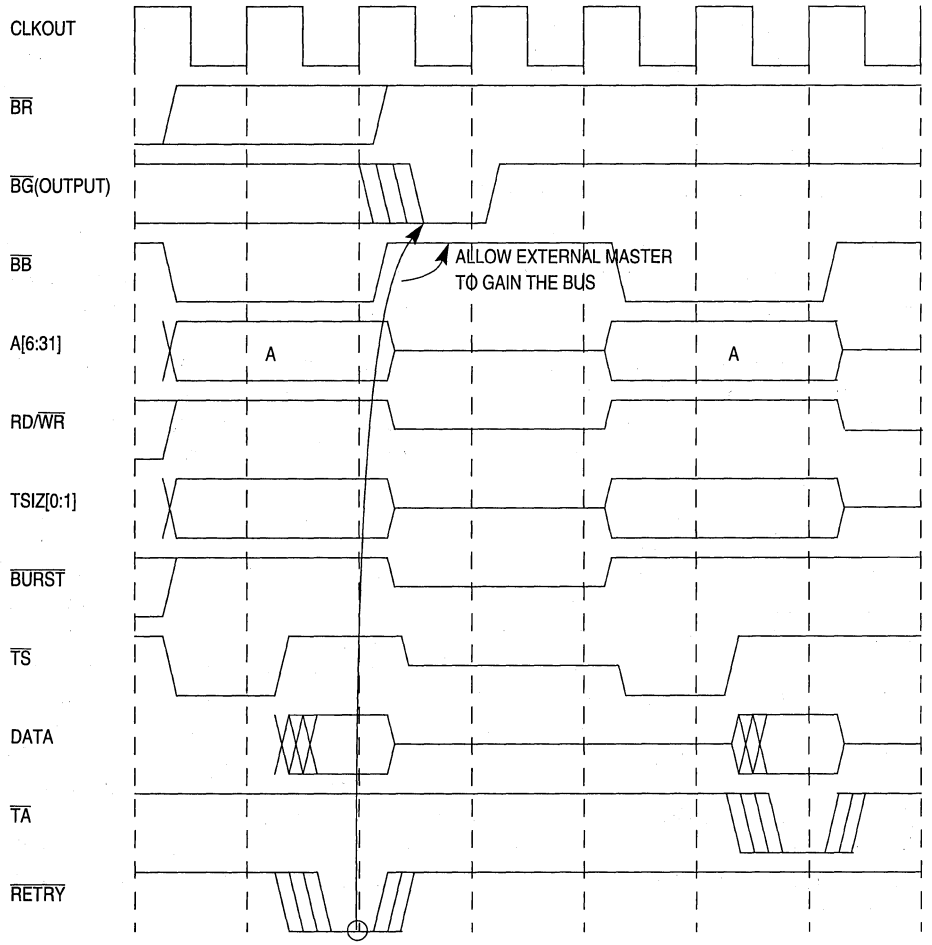


Figure 13-28. RETRY Transfer Timing–Internal Arbiter

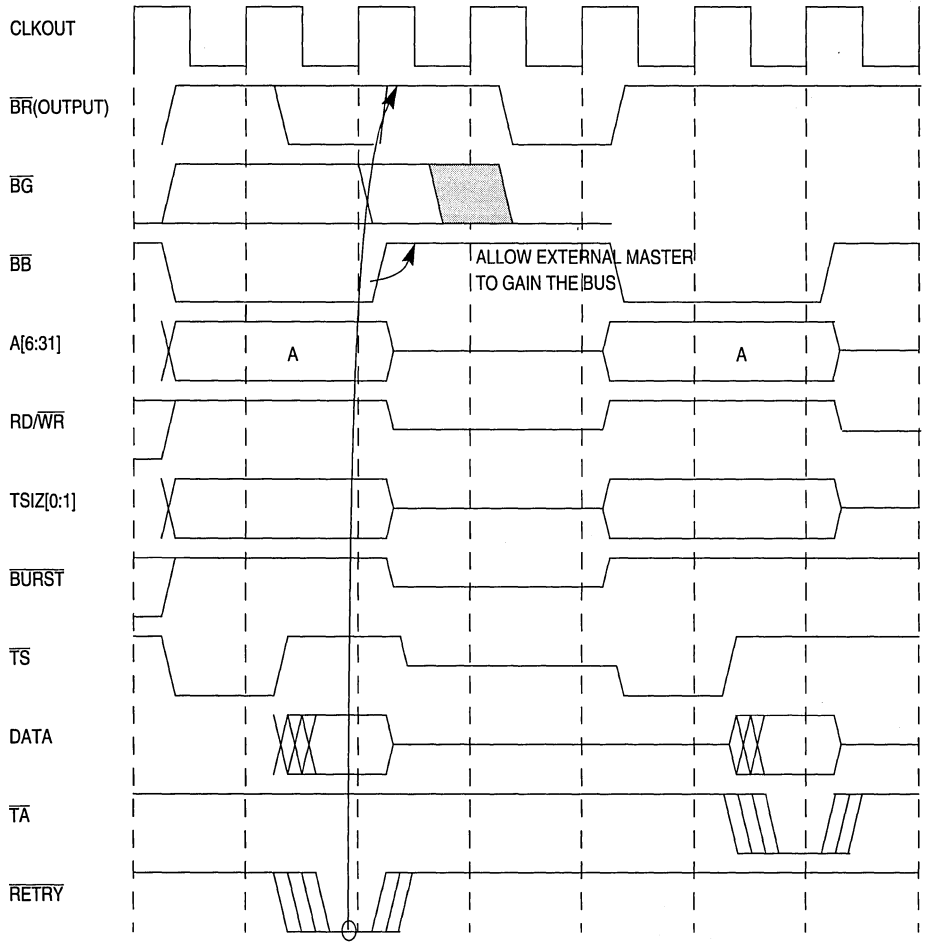


Figure 13-29. RETRY Transfer Timing–External Arbiter

When a burst access is initiated by the MPC823, the bus interface only recognizes the RETRY assertion as a retry termination if it detects it before the first data beat is acknowledged by the slave device. When the RETRY signal is asserted as a termination signal on the second or third data beat of the access, the MPC823 recognizes it as a transfer error acknowledgement.

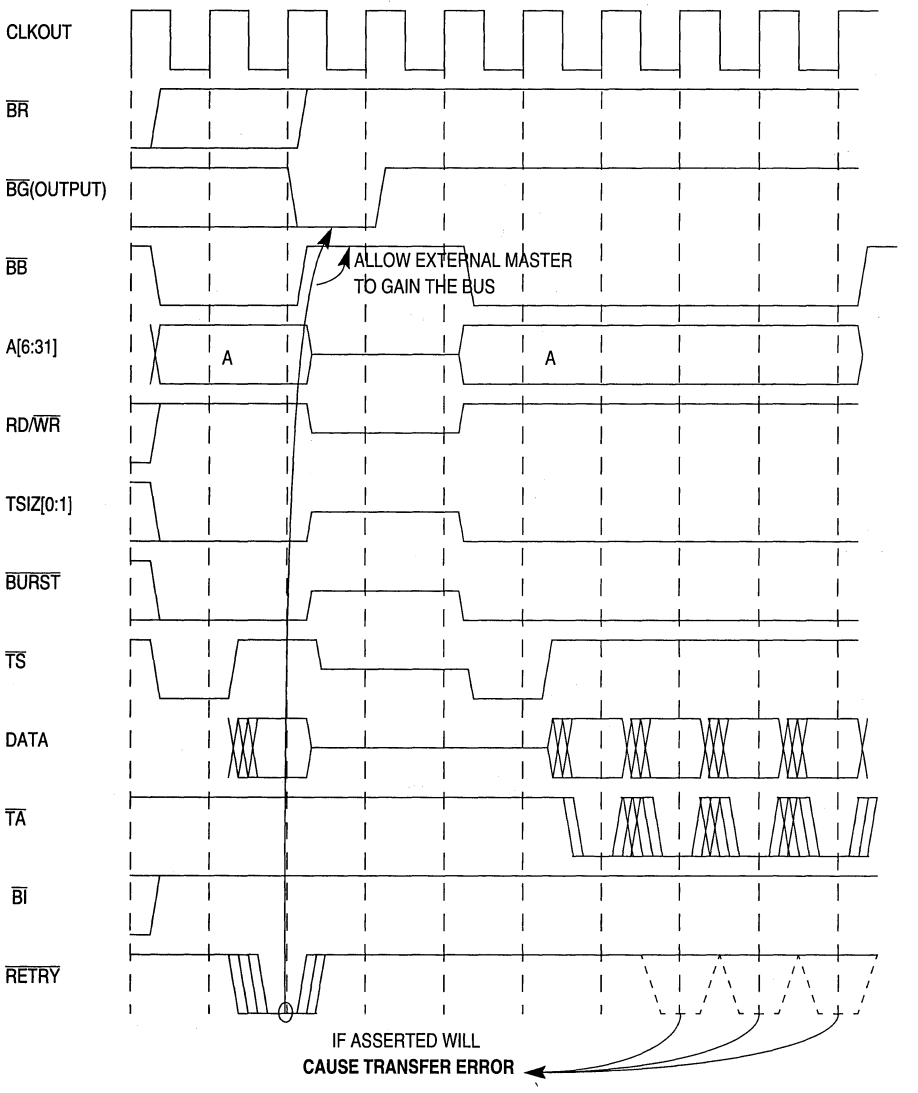


Figure 13-30. Retry On Burst Cycle

In reference to Figure 13-30, if the $\overline{BI}$ signal is asserted at the first beat of a burst, then the remaining beats of the 16-byte transfer retry are recognized as a transfer error acknowledge. Table 13-6 summarizes how the MPC823 recognizes the termination signals provided by the slave device that the initiated transfer addressed.

Table 13-6. Termination Signal Protocol

$\overline{TEA}$	$\overline{TA}$	$\overline{RETRY}/\overline{KR}$	ACTION
Asserted	X	X	Transfer Error Termination
Negated	Asserted	X	Normal Transfer Termination
Negated	Negated	Asserted	Retry Transfer Termination / Kill Reservation

SECTION 14
ENDIAN MODES

This section will discuss how the MPC823 supports three system endian configurations:

- Little-endian system
- Big-endian system
- PowerPC™ little-endian system

A general description of the different endian modes can be found in *The PowerPC Microprocessor Family: The Programming Environments* (MPCFPE/AD) manual that is available from Motorola. Throughout the *MPC823 User's Manual*, the term *system* refers to the devices that reside on the MPC823 bus. The MPC823 core operates in the big-endian mode of a big-endian system and in the PowerPC little-endian mode of two other configurations.

Table 14-1. Little-Endian Effective Address Modification For Individual Aligned Scalar

DATA LENGTH (BYTES)	ADDRESS MODIFICATION:
1	XOR with 0b111
2	XOR with 0b110
4	XOR with 0b100
8	(No Change)

NOTE: There are no 8-byte scalars in the MPC823.

For programming configurations, refer to the table below.

Table 14-2. Endian Mode Programming For Core Data Structures

MODE	MSR_{LE} (AND MSR_{ILE})	DC_CSR_{LES}
Big-Endian Mode	0	0
Little-Endian Mode	0	1
PowerPC Little-Endian Mode	1	0
Reserved	1	1

The hardware operations that are used to support the different endian modes are:

- Address munging in the core is controlled by the MSR_{LE} bit. Refer to the *PowerPC Microprocessor Family: The Programming Environments for 32-Bit Microprocessors* manual for more information.
- The MPC823 internal bus signal is driven by the master that informs the system interface unit to swap and perform address demunging or leave the current access as it is. Table 10-1 defines the DC_CSR_{LES} bit for core and cache accesses.
- Address munging and data bytes format in the communication processor module (CPM) that is controlled by the BO field of the function code register.

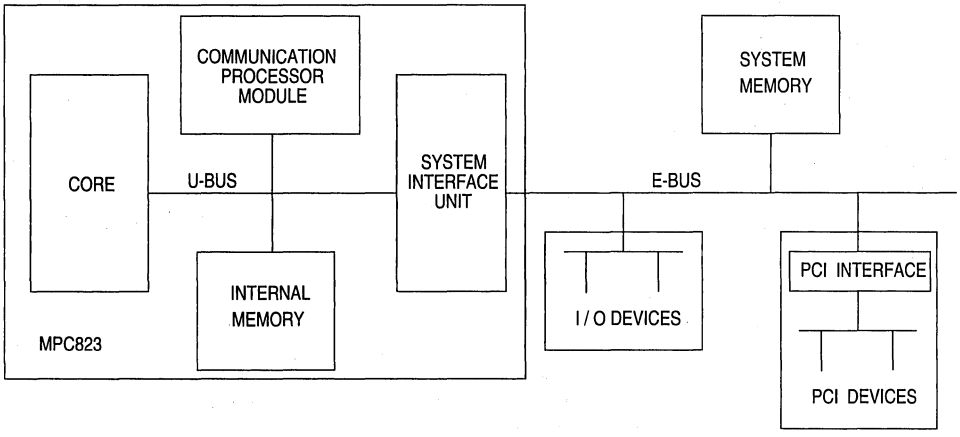


Figure 14-1. General MPC823 System Diagram

Note: Some peripheral component interconnect (PCI) bridge devices (such as the MPC105/106) cannot be used in little-endian mode.

14.1 LITTLE-ENDIAN FEATURES

The following is a list of the little-endian system's main features.

- System Memory Organization and E-Bus Format are Little-Endian
- U-Bus Data, Instruction and Data Caches, and Internal Memory Format are Big-Endian
- Data Access Constraints That Follow the PowerPC Little-Endian Rules
- Same Byte Order Between the Media and System Memory
- For Core Accesses, Swap and Address Demunging Are Performed By the System Interface Unit On the U-Bus To the System Path
- The Core's Load/Store Unit Swapper Uses Munged Addresses to Put the Data On the Right Byte Lanes When Half-Word or Byte Accesses Are Performed
- The Communication Processor Module Performs Data Swapping According to Information in Buffer Descriptors

The following tables describe how to handle the little-endian program or data in the little-endian system that is built around the MPC823 for various port sizes.

Table 14-3. Little-Endian Program/Data Path Between the Register and 32-Bit Memory

FETCH/ LOAD STORE TYPE	LITTLE-ENDIAN ADDR	U-BUS AND CACHES ADDR	EXTERNAL BUS ADDR	DATA IN THE REGISTER				U-BUS AND CACHES FORMAT				E-BUS FORMAT				LITTLE-ENDIAN PROGRAM/DATA			
				0	1	2	3	0	1	2	3	0	1	2	3	3	2	1	0
Word	0	0	0	11	12	13	14	11	12	13	14	14	13	12	11	11	12	13	14
Half-word	0	2	0			21	22			21	22	22	21					21	22
Half-word	2	0	2			31	32	31	32					32	31	31	32		
Byte	0	3	0				'a'				'a'	'a'							'a'
Byte	1	2	1				'b'			'b'			'b'					'b'	
Byte	2	1	2				'c'		'c'					'c'			'c'		
Byte	3	0	3				'd'	'd'							'd'	'd'			

Table 14-4. Little-Endian Program/Data Path Between the Register and 16-Bit Memory

FETCH/ LOAD STORE TYPE	LITTLE-ENDIAN ADDR	U-BUS AND CACHES ADDR	EXTERNAL BUS ADDR	DATA IN THE REGISTER				U-BUS AND CACHES FORMAT				E-BUS FORMAT				LITTLE-ENDIAN PROGRAM/DATA			
				0	1	2	3	0	1	2	3	0	1	2	3	3	2	1	0
Word	0	0	0	11	12	13	14	11	12	13	14	14	13					13	14
			2									12	11					11	12
Half-word	0	2	0			21	22			21	22	22	21					21	22
Half-word	2	0	2			31	32	31	32			32	31					31	32
Byte	0	3	0				'a'				'a'	'a'							'a'
Byte	1	2	1				'b'			'b'			'b'					'b'	
Byte	2	1	2				'c'		'c'			'c'							'c'
Byte	3	0	3				'd'	'd'					'd'					'd'	

Table 14-5. Little-Endian Program/Data Path Between the Register and 8-Bit Memory

FETCH/ LOAD STORE TYPE	LITTLE-ENDIAN ADDR	U-BUS AND CACHES ADDR	EXTERNAL BUS ADDR	DATA IN THE REGISTER				U-BUS AND CACHES FORMAT				E-BUS FORMAT				LITTLE-ENDIAN PROGRAM/DATA			
				0	1	2	3	0	1	2	3	0	1	2	3	3	2	1	0
Word	0	0	0	11	12	13	14	11	12	13	14	14							14
			1									13							13
			2									12							12
			3									11							11
Half-word	0	2	0			21	22			21	22	22							22
			1									21							21
Half-word	2	0	2			31	32	31	32			32							32
			3									31							31
Byte	0	3	0				'a'				'a'	'a'							'a'
Byte	1	2	1				'b'			'b'		'b'							'b'
Byte	2	1	2				'c'		'c'			'c'							'c'
Byte	3	0	3				'd'	'd'				'd'							'd'

14.2 BIG-ENDIAN SYSTEM FEATURES

The following is a list of the big-endian system's main features:

- Caches, U-Bus, E-Bus, System Memory, and I/O Organization Format is Big-Endian
- Same Byte Order Between the Media and System Memory
- Communication Processor Module Writes and Reads Big-Endian U-Bus Data
- PCI Bridge Operates in Big-Endian Mode as Needed

14.3 POWERPC LITTLE-ENDIAN SYSTEM FEATURES

The following is a list of the PowerPC little-endian system's main features:

- Caches, U-Bus, E-Bus, System Memory, and E-Bus Attached I/O Organization Format is Big-Endian
- PCI Bus Format is Little-Endian
- Data Access Constraints That Follow the PowerPC Little-Endian Rules
- Address Munging in the Core and Communication Processor Module Follows the Guidelines in Table 14-1
- The PCI Bridge Operates in Little-Endian Mode as Needed. Swap and Address Demunging is Performed by the PCI Bridge on the PCI I/O to the System Memory Path.
- The Stream Hit Mechanisms of the Instruction and Data Caches Operate Less Efficiently When Address Munging is Performed on Cache Accesses. Some Performance Degradation is Expected When Working in this Mode.

14.4 SETTING THE ENDIAN MODE OF OPERATION

The mode should be set early in the reset routine and remain unchanged for the duration of system operation. The MPC823 core is in big-endian mode after reset. To switch between the different endian modes of operation, the core must run in serialized mode and the caches should be disabled. It is not recommended that you switch back and forth between modes. To transfer the system to the PowerPC little-endian mode, the MSR_{LE} and MSR_{ILE} bits should be changed with the **mtmsr** instruction that resides on the odd word boundary (A[29] = 1). The instruction that is executed next will be fetched from this address plus 8. If the instruction resides on an even word boundary (A[29] = 0), then this instruction will be executed twice because of address munging.

The instruction used to transfer the system back to big-endian mode must reside on an even word boundary (A[29] = 0). The next instruction will be fetched from this address plus 12. A transfer to little-endian mode should be made with the **mtspr** instruction that resides on the even word boundary (A[29] = 0). Further instructions should reside in the little-endian format of the external system memory and in the big-endian format of the internal memory. The BO field of the function code registers (FCRs) in the communication processor module should be set to the required endian format for the buffer descriptor.

SECTION 15
MEMORY CONTROLLER

The memory controller is responsible for controlling a maximum of eight memory banks shared between a general-purpose chip-select machine and a pair of sophisticated user-programmable machines. It supports a glueless interface to SRAM, EPROM, flash EPROM, regular DRAM devices, self-refresh DRAMs, extended data output DRAM devices, synchronous DRAMs, and other peripherals. This flexible memory controller allows you to implement memory systems with very specific timing requirements. It supports external address multiplexing, periodic timers, and timing generation for row address and column address strobes to allow for a glueless interface to DRAM devices. The periodic timers allow refresh cycles to be initiated while the address muxing provides row and column addresses.

You can define different timing patterns for the control signals that govern a memory device. These patterns define how the external control signals behave in read-access, write-access, burst read-access, or burst write-access requests. You decide how the external control signals toggle when the periodic timers reach the maximum programmed value for refresh operation.

15.1 FEATURES

The following is a list of the memory controller's main features:

- Eight Memory Banks
  - ❑ 32-bit address decode with mask
  - ❑ Various block sizes (32K to 4G)
  - ❑ Byte parity generation/checking
  - ❑ Write-protection capability
  - ❑ "Address types" match qualifying memory bank accesses for internal masters
  - ❑ Timing pattern machine selected according to the type of memory device accessed
  - ❑ Support for external master access to memory banks
  - ❑ Synchronous and asynchronous external masters support
- General-Purpose Chip-Select Machine
  - ❑ Compatible with SRAM, EPROM, FEPROM, and peripherals
  - ❑ Global (boot) chip-select available at system reset
  - ❑ Boot chip-select support for 8-, 16-, and 32-bit devices
  - ❑ Two clock accesses to external device
  - ❑ Four byte write enable ($\overline{WE}$[0:3]) signals
  - ❑ Output enable ($\overline{OE}$) signal

- Two User-Programmable Machines
  - ❑ RAM-based machine controls the timing of the external signals with a granularity of one quarter of a system clock period
  - ❑ User-specified patterns run when a single read access, single write access, burst read access or burst write access is requested by an internal or external synchronous master
  - ❑ User-specified patterns run when a single read access or single write access is requested by an external asynchronous master
  - ❑ UPM periodic timer initiates an automatic pattern when it expires (refresh)
  - ❑ User-specified patterns run under software control
  - ❑ Each UPM can be defined to support DRAM devices with depths of 64K,128K, 256K, 512K, 1M, 2M, 4M, 8M, 16M, 32M, 64M, 128M, and 256M
  - ❑ Four byte-select lines for each UPM
  - ❑ Six external general-purpose lines controlled by each UPM
  - ❑ Supports 8-, 16-, and 32-bit DRAM port sizes
  - ❑ Glueless interface to one bank of DRAM (only external buffers are required for additional SIMM banks)
  - ❑ Page mode support for successive transfers within a burst for all on-chip and external synchronous masters
  - ❑ Internal address multiplexing for all on-chip bus masters supporting 64K, 128K, 256K, 512K, 1M, 2M, 4M, 8M, 16M, 32M, 64M, 128M, 256M page banks
  - ❑ Glueless interface to EDO, self refresh, and synchronous DRAM devices

A block diagram of the memory controller is illustrated in Figure 15-1.

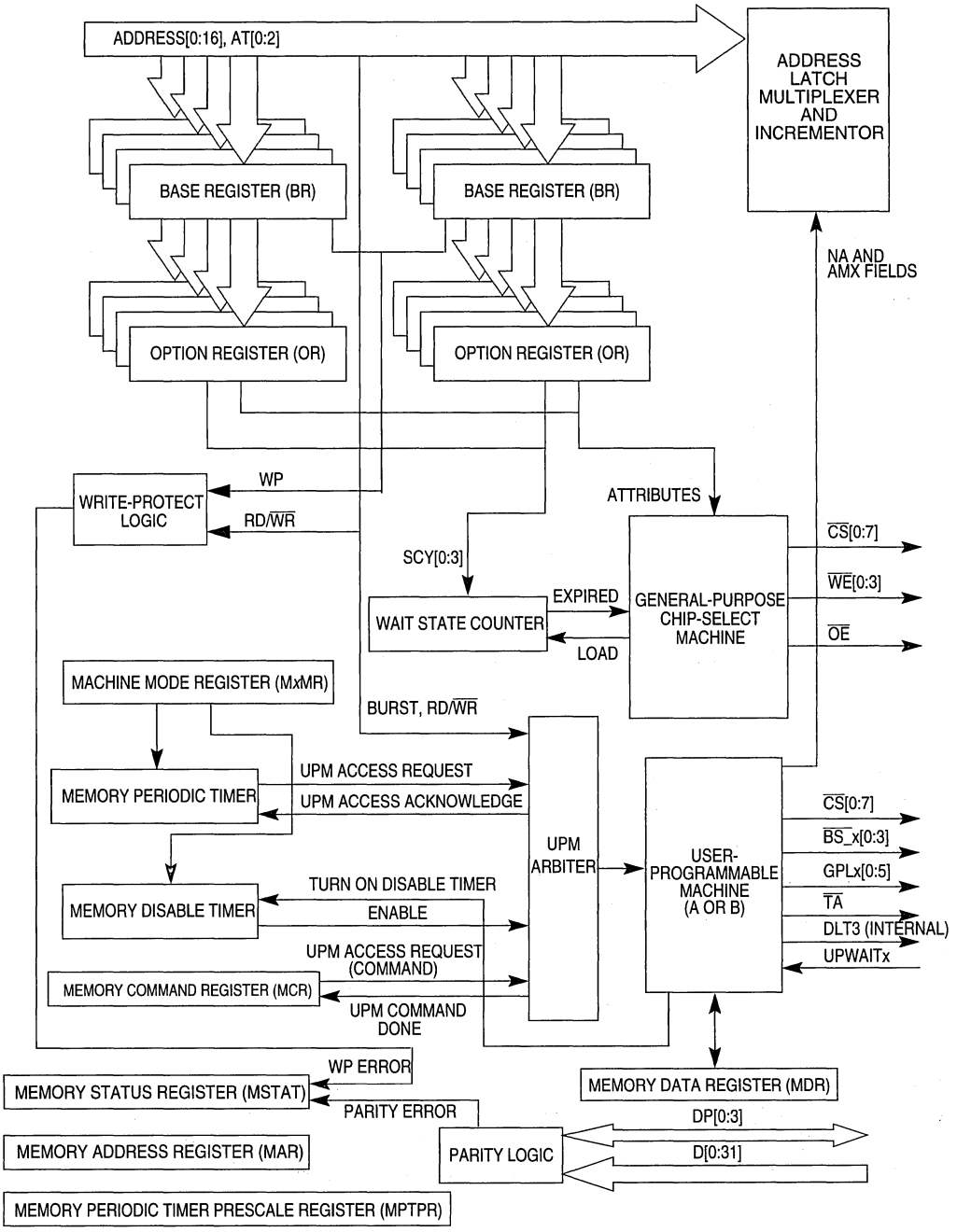


Figure 15-1. Memory Controller Block Diagram (Single UPM)

15.2 ARCHITECTURE

The memory controller consists of three basic machines:

- General-purpose chip-select machine
- User-programmable machine A
- User-programmable machine B

As illustrated in Figure 15-2, each bank can be assigned to any one of these machines via the MS field in the base register. When a memory address matches the BA field of the base register, the corresponding machine takes ownership of the external signals that control access until the cycle terminates.

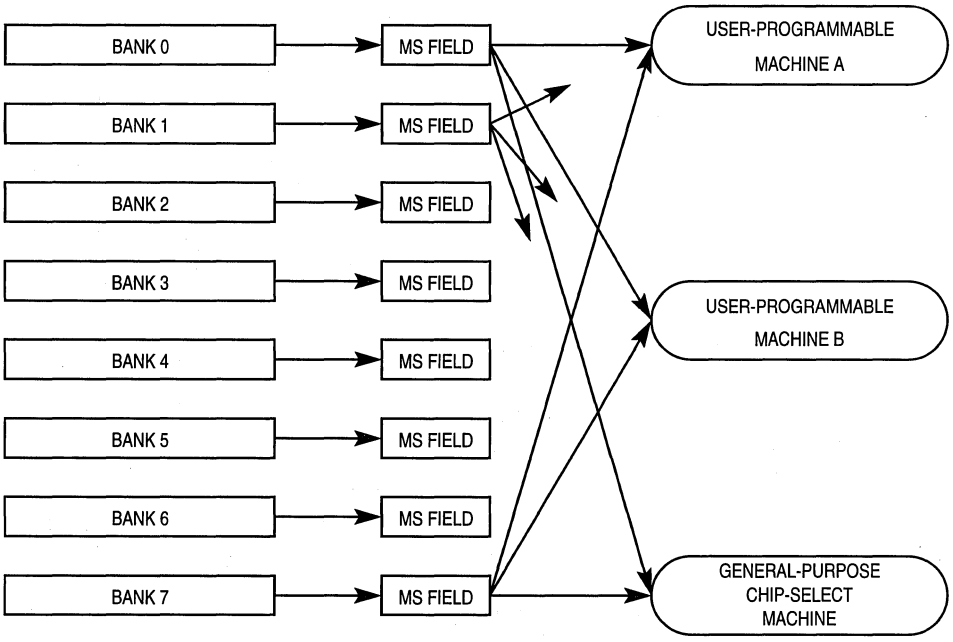


Figure 15-2. Memory Controller Machine Selection

The general-purpose chip-select machine (GPCM) provides a glueless interface to EPROM, SRAM, Flash EPROM, and other peripherals. General-purpose chip-select signals are available on $\overline{CS}$[0:7]. $\overline{CS0}$ also functions as the boot chip-select signal that allows the CPU to access the boot EPROM from reset. Each chip-select allows a maximum of 30 wait states.

Some features are common to all eight memory banks. The full 32-bit decode is available internally, even if all 32 address bits are not visible outside the MPC823. For external master transactions, the memory controller extends the 26-bit external address line to 32 bits and the six most-significant bits are zero. The variable block size of each memory bank can be between 32K and 64M for a total memory capacity of 512M. Parity can be generated and checked for any memory bank and each memory bank can be selected for read-only or read/write operation. For system protection purposes, you can use certain address type codes to cause the memory controller to restrict access to a memory bank. For additional flexibility, address type comparisons provide you with a mask option.

The memory controller functionality helps you design MPC823-based systems with little or no glue logic required. In Figure 15-3, $\overline{CS0}$ is used as the 16-bit boot EPROM with the MS field of the base register 0 (BR0) configured to select the GPCM. $\overline{CS1}$ is used as the $\overline{RAS}$ signal for 32-bit DRAM with the MS field of base register 1 (BR1) configured to select UPMA. The $\overline{BS\_A}$ signals are used as $\overline{CAS}$ signals on the DRAM.

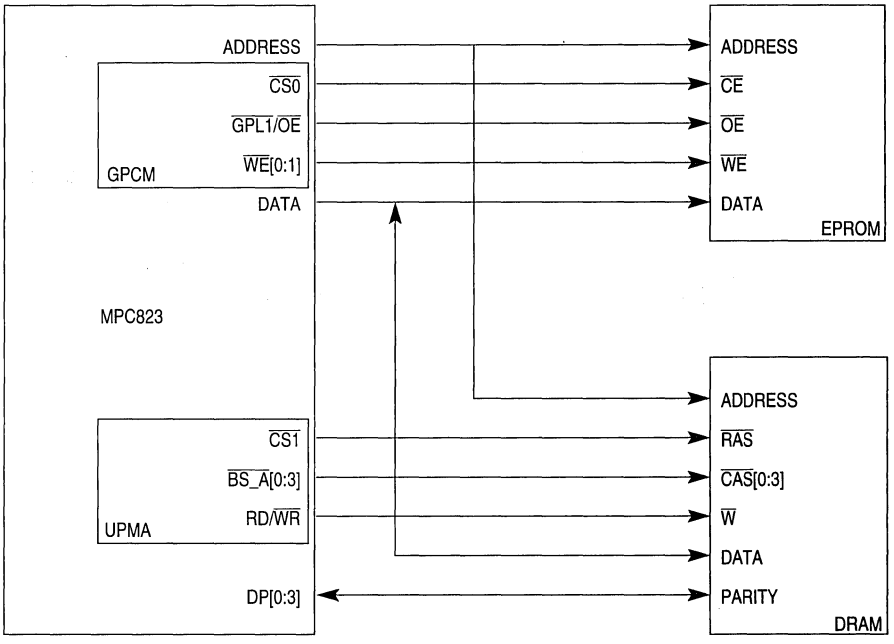


Figure 15-3. Simple System Configuration

The two user-programmable machines (UPMA and UPMB) in the memory controller provide a flexible interface to many types of memory devices. Each UPM can control the address multiplexing necessary to access DRAM devices, the timing of the $\overline{BS}$ signals, and the timing of the $\overline{GPLx}$ signals. Each memory bank can be assigned to either UPM. There are a total of eight $\overline{CSx}$ signals that can be split between the two UPMs.

Each user-programmable machine is a RAM-based machine controlled by software. The software toggles the memory controller external signals when an external single word read/write access or an external burst read/write access is initiated by an internal or external master. The UPM also controls address multiplexing, address increment, and transfer acknowledge assertion for a specific memory access. The UPM can be programmed to run a specific signal pattern for a certain duration of clock cycles. At every clock cycle, the logical value of the external signals specified in the RAM array is output on the corresponding UPM pins.

When a new access to external memory is requested by any of the internal or external masters, the address of the transfer and the address type is compared to each one of the valid banks defined in the memory controller. Notice that all of the A[0:16] and AT[0:2] signals are maskable. When an address match is found in one of the memory bank chip-select ranges, the corresponding MS field in the base register defines the machine that handles the memory access. See Figure 15-4 for details.

The memory controller provides four parity (DP[0:3]) signals, one for each data byte lane on the MPC823 system bus. The parity on the bus is only checked if the memory bank accessed in the current transaction has parity enabled. Parity checking/generation can be enabled for a specific memory bank in the base register. The type of parity is defined in the system interface unit module configuration register, which is explained in **Section 12.12.1.1 SIU Module Configuration Register**. Also, system protection is provided by defining each memory bank as read-only or read/write.

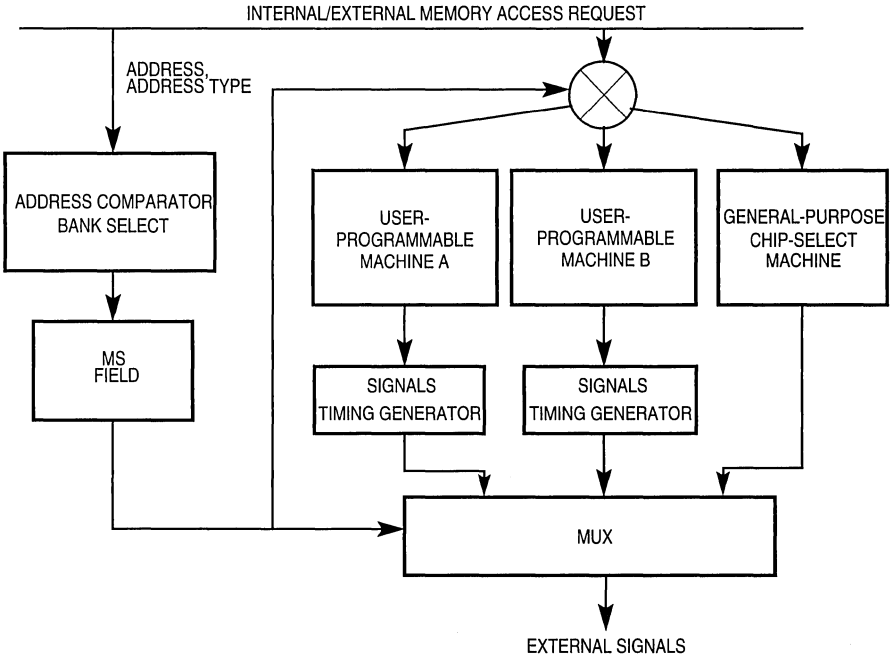


Figure 15-4. Basic Memory Controller Operation

15.3 REGISTER MODEL

The status bits for each one of the memory banks are in the memory controller status (MSTAT) register, which is used by the entire memory controller. Each of the eight memory banks has a base register (BRx) and an option register (ORx). The MSTAT reports write-protect violations that occur and parity errors for every bank. The base register contains a V bit that indicates when the information for the chip-select is valid.

Each base register defines the starting address of its memory bank and each option register defines the attributes for its memory bank. The option registers also define the initial address multiplexing for a memory cycle controlled by a UPM. The machine A mode register (MAMR) and machine B mode register (MBMR) define most of the global features for the user-programmable machines.

The memory command register (MCR) and memory data register (MDR) are used to initialize the UPM's RAM array. The memory address register (MAR) specifies the location in the RAM array to be executed as defined by the MCR. It also allows a specific address pattern to be output onto the A[6:31] signals. The memory periodic timer prescaler register (MPTPR) defines the divisor of the BRGCLK used as the memory periodic timer input clock.

The memory controller registers are used by the general-purpose chip-select machine and the user-programmable machines as specified in Table 15-1. See **Section 15.3.1 Register Descriptions** for specific register information.

Table 15-1. Memory Controller Register Usage

REGISTER	USED BY THE GPCM	USED BY A UPM
Base Register Bank 0-7 Register (BRx)	√	√
Option Register Bank 0-7 Register (ORx)	√	√
Memory Status Register (MSTAT)	√	√
Memory Command Register (MCR)		√
Machine A Mode Register (MAMR)		√
Machine B Mode Register (MBMR)		√
Memory Data Register (MDR)		√
Memory Address Register (MAR)		√
Memory Periodic Timer Prescaler Register (MPTPR)		√

The memory controller supports multiple port sizes. Predefined 8-bit ports can be accessed as odd or even bytes, predefined 16-bit ports can be accessed as odd or even bytes and even half-words on data bus bits 0 through 15. Predefined 32-bit ports can be accessed as odd bytes, even bytes, odd half-words, even half-words, or words on word boundaries. The port size is specified by the PS field in the base register.

The WP bit of the base register restricts write accesses to a certain address range. If you try to write in this area, a write-protect violation occurs and the WPER bit in the memory status register is set.

Each time an internal or external bus cycle access is requested, the address and its corresponding address type are compared to each one of the banks. If a match is found on one of the memory controller banks, the attributes defined for that bank in the base and option registers are used to control the memory access. However, if multiple matches are found, the lowest numbered matched bank handles the memory access. It should be noted that when external masters access memory controller-managed slaves on the bus, the internal AT signals to the memory controller are forced to '100'.

Parity can be configured for any bank. It is generated and checked on a per-byte basis using the DP[0:3] signals for the bank if the PARE bit is set in the base register. The OPAR bit in the system interface unit module configuration register determines the type of parity, as described in **Section 12.12.1.1 SIU Module Configuration Register**. Any parity error causes the associated PER bit in the memory status register to be set. It also asserts the $\overline{\text{TEA}}$ signal and sets the corresponding DPB bit in the transfer error status register (TESR), which is described in **Section 12.12.1.4 Transfer Error Status Register**. The memory controller asserts an internal transfer error signal when a parity error occurs (if enabled).

15.3.1 Register Descriptions

15.3.1.1 BASE REGISTERS. The base registers (BR0-7) contain the base address and address types that are used by the memory controller to compare the address bus with the current address accessed. It also includes a memory attribute and selects the machine for memory operation handling. After reset, BR0 is referred to as the Boot BR0 and it has a special functionality until the first write to OR0.

BOOT BR0

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	BA															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x100															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	BA	AT			PS		PARE	WP	MS		RESERVED					V
RESET	0	0			*		0	0	0		0					*
R/W	R/W	R/W			R/W		R/W	R/W	R/W		R/W					R/W
ADDR	(IMMR & 0xFFFF0000) + 0x102															

* This value depends on the value of the hard reset configuration word.

BR*x*

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	BA															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x100 (BR0), 0x108 (BR1), 0x110, (BR2), 0x118 (BR3), 0x120 (BR4), 0x128 (BR5), 0x130 (BR6), 0x138 (BR7)															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	BA	AT			PS		PARE	WP	MS		RESERVED					V
RESET	0	0			0		0	0	0		0					0
R/W	R/W	R/W			R/W		R/W	R/W	R/W		R/W					R/W
ADDR	(IMMR & 0xFFFF0000) + 0x102 (BR0), 0x10A (BR1), 0x112, (BR2), 0x11A (BR3), 0x122 (BR4), 0x12A (BR5), 0x132 (BR6), 0x13A (BR7)															

BA—Base Address

This field, the upper 17 bits of each base address register, and the AT field are compared to the address on the address bus to determine if a memory bank controlled by the memory

controller is being accessed by an internal bus master. These bits are used in conjunction with the AM field of the option register.

AT—Address Type

This field can be used to limit accesses to the memory bank to a certain address space type. These bits are used in conjunction with the ATM field of the option register.

PS—Port Size

This field specifies the port size of the memory region. After system reset, the value of this bit in BR0 depends on the BPS field value in the hard reset configuration word. Refer to **Section 4.3.1.1 Hard Reset Configuration Word** for more information.

00 = 32-bit port size.
01 = 8-bit port size.
10 = 16-bit port size.
11 = Reserved.

PARE—Parity Enable

This bit is used to enable parity checking on this bank.

0 = Parity checking is disabled.
1 = Parity checking is enabled.

WP—Write-Protect

This bit may restrict write accesses within the address range of a base register. If you try to write to the range of addresses specified in a base address register that has this bit set, the bus monitor logic asserts the $\overline{TEA}$ signal which then terminates the cycle.

0 = Both read and write accesses are allowed.
1 = Only read accesses are allowed. The $\overline{CSx}$ and $\overline{TA}$ signals are not asserted by the memory controller on write cycles to this memory bank. The WPER bit is set in the MSTAT register if you try to write to this memory bank.

MS—Machine Select

This field specifies the machine that is selected for memory operations handling.

00 = GPCM.
01 = Reserved.
10 = UPMA.
11 = UPMB.

Bits 26–30—Reserved

These bits are reserved and should be set to 0.

V—Valid

This bit indicates that the contents of the base and option registers are valid. The $\overline{CSx}$ signal does not assert until this bit is set. An access to a region that does not have this bit set can cause a bus monitor timeout. After a system reset, the value of this bit in BR0 depends on the BDIS bit value in the hard reset configuration word, which is described in **Section 4.3.1.1 Hard Reset Configuration Word**.

0 = This bank is invalid.
1 = This bank is valid.

15.3.1.2 OPTION REGISTERS. The option registers (OR0-7) contain the address mask and address type mask bit for address bus comparison. It also includes the CS general field and all the GPCM parameters. After reset, OR0 is referred to as the Boot OR0 and it has a special functionality until the first write to OR0.

BOOT OR0

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	AM															
RESET	0															
R/W	R															
ADDR	(IMMR & 0xFFFF0000) + 0x104															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	AM	ATM			CSNT/ SAM	ACS/G5LA,G5LS		BIH	SCY				SETA	TRLX	EHTR	RES
RESET	0	0			1	11		1	1				0	1	0	0
R/W	R	R			R	R		R	R				R	R	R	R
ADDR	(IMMR & 0xFFFF0000) + 0x106															

OR*x*

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	AM															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x104 (OR0), 0x10C (OR1), 0x114 (OR2), 0x11C (OR3), 0x124 (OR4),0x12C (OR5), 0x134 (OR6), 0x13C (OR7)															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	AM	ATM			CSNT/ SAM	ACS/G5LA,G5LS		BIH	SCY				SETA	TRLX	EHTR	RES
RESET	0	0			0	0		0	0				0	0	0	0
R/W	R/W	R/W			R/W	R/W		R/W	R/W				R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x106 (OR0), 0x10E (OR1), 0x116 (OR2), 0x11E (OR3), 0x126 (OR4), 0x12E (OR5), 0x136 (OR6), 0x13E (OR7)															

NOTE: The reset value of OR0 has predefined values as shown in the boot OR0 register table.

AM—Address Mask

This read/write field provides masking on any corresponding bits in the associated base register. By masking the address bits independently, external devices of different size address ranges can be used. Any cleared bit masks the corresponding address bit and any set bit causes the corresponding address bit to be used in address pin comparison. The AM field can be set or cleared in any order in the field, thus allowing a resource to reside in more than one area of the address map.

ATM—Address Type Mask

This field masks certain bits in an address type, thus allowing more than one address space type to be assigned to a chip-select. Any set bit causes the corresponding address type code bits to be used as part of the address comparison. Any cleared bit masks the corresponding address type code bit. The ATM field should be cleared so that address type codes are ignored as part of the address comparison.

CSNT—Chip-Select Negation Time/SAM—Start Address Multiplex

This bit is used for the GPCM and the SAM bit is used for the UPM. The CSNT bit, in conjunction with ACS and TRLX, is used to control negation of the $\overline{CSx}$ and $\overline{WEx}$ signals during an external memory write access handled by the general-purpose chip-select machine. This function provides extended address/data hold time for slower memories and peripherals. See Table 15-2 (page 15-29) for more information.

The SAM bit determines the address output on the first cycle of an external memory access.

0 = Address pins reflect the address requested by the internal master.
1 = Address pins reflect the address requested by the internal master multiplexed according to the AMA field (if UPMA is selected to control the memory access) or the AMB field (if UPMB is selected).

ACS—Address to Chip-Select Setup/G5LA

This field is used for the GPCM and the G5LA and G5LS fields are used for the UPM. This field controls $\overline{CSx}$ signal assertion in relation to address lines valid.

00 = $\overline{CS}$ is output at the same time as the address lines.
01 = Reserved.
10 = $\overline{CS}$ is output a quarter of a clock later than the address lines.
11 = $\overline{CS}$ is output half a clock later than the address lines.

G5LS—General-Purpose Line 5 A

This field determines how the internal timing generator ($\overline{GPL5}$) signal is output when the memory access is handled by the UPMA or UPMB.

G5LA (only valid for UPMB):
0 = Output the internal $\overline{GPL5}$ signal on the $\overline{GPL\_B5}$ pin.
1 = Output the internal $\overline{GPL5}$ signal on the $\overline{GPL\_A5}$ pin.

G5LS (valid for UPMA or UPMB):
0 = The $\overline{GPL5}$ signal is driven low on the falling edge of GCLK1 during the first clock cycle of a read or write memory access.
1 = The $\overline{GPL5}$ signal is driven high on the falling edge of GCLK1 during the first clock cycle of a read or write memory access.

BIH—Burst Inhibit

This bit determines whether or not this memory bank supports burst accesses. When a burst does not occur, the memory controller drives the $\overline{BI}$ signal active when accessing this memory region. If the machine selected to handle this access is the GPCM, this bit should be set to 1.

0 = The $\overline{BI}$ pin is negated. The bank supports burst accesses.
1 = The $\overline{BI}$ pin is asserted. The bank does not support burst accesses.

SCY—Select Cycle Length (GPCM only)

This field determines the number of wait states inserted in the cycle when the general-purpose chip-select machine handles the external memory access. It is one of the parameters that control the cycle's length. The total cycle length is controlled by this parameter and the TRLX field. Refer to Table 15-2 (page 15-29) for the total number of cycles.

If you want to use an external $\overline{TA}$ response (SETA bit = 1), then these bits are not used.

0000 = 0 clock cycle wait state.
0001 = 1 clock cycle wait state.
0010 = 2 clock cycle wait states.
0011 = 3 clock cycle wait states.
0100 = 4 clock cycle wait states.
0101 = 5 clock cycle wait states.
0110 = 6 clock cycle wait states.
0111 = 7 clock cycle wait states.
1000 = 8 clock cycle wait states.
1001 = 9 clock cycle wait states.
1010 = 10 clock cycle wait states.
1011 = 11 clock cycle wait states.
1100 = 12 clock cycle wait states.
1101 = 13 clock cycle wait states.
1110 = 14 clock cycle wait states.
1111 = 15 clock cycle wait states.

SETA—Select External Transfer Acknowledge (GPCM only)

This bit indicates when the $\overline{TA}$ signal is externally generated once the GPCM is selected to handle the memory access that was initiated to this memory region. Regardless of other setup parameters for the GPCM, if SETA = 1, then all control signals of the memory controller are negated after the externally generated $\overline{TA}$ signal is recognized.

0 = Internal or external transfer acknowledge can acknowledge this memory access, whichever comes first.
1 = Transfer acknowledge must be provided by external logic.

TRLX—Timing Relaxed (GPCM only)

When this bit is set, it extends the timing of the signals controlling the memory devices once the GPCM is selected to handle the memory access that was initiated to this memory region. Refer to Table 15-2 (page 15-29) for more information.

0 = Timing is defined by the GPCM.
1 = Relaxed timing is defined by the GPCM.

EHTR—Extended Hold Time on Read

When this bit is set, it adds one clock cycle after a read from the current bank and any CPU write or read to a different bank.

0 = Timing is defined by the memory controller.
1 = Extended hold time is defined on the current read access.

Bit 31—Reserved

This bit is reserved and should be set to 0.

15.3.1.3 MEMORY STATUS REGISTER. The memory status (MSTAT) register reports parity and write-protect errors encountered during an external bus access initiated by the memory controller. To clear a specific bit, write a one to it (writing zero has no effect).

MSTAT

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PER0	PER1	PER2	PER3	PER4	PER5	PER6	PER7	WPER	RESERVED						
RESET	0	0	0	0	0	0	0	0	0	0						
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W						
ADDR	(IMMR & 0xFFFF0000) + 0x178															

PER0—Parity Error Bank 0

When this bit is set it indicates that a parity error was detected during a Bank 0 read cycle initiated by the memory controller.

PER1—Parity Error Bank 1

When this bit is set it indicates that a parity error was detected during a Bank 1 read cycle initiated by the memory controller.

PER2—Parity Error Bank 2

When this bit is set it indicates that a parity error was detected during a Bank 2 read cycle initiated by the memory controller.

PER3—Parity Error Bank 3

When this bit is set it indicates that a parity error was detected during a Bank 3 read cycle initiated by the memory controller.

PER4—Parity Error Bank 4

When this bit is set it indicates that a parity error was detected during a Bank 4 read cycle initiated by the memory controller.

PER5—Parity Error Bank 5

When this bit is set it indicates that a parity error was detected during a Bank 5 read cycle initiated by the memory controller.

PER6—Parity Error Bank 6

When this bit is set it indicates that a parity error was detected during a Bank 6 read cycle initiated by the memory controller.

PER7—Parity Error Bank 7

When this bit is set it indicates that a parity error was detected during a Bank 7 read cycle initiated by the memory controller.

WPER—Write-Protection Error

This bit is set when a write-protect error has occurred on a write cycle to a write-protected bank. The write-protect error is also stored in the transfer error status register of the system interface unit. See **Section 12.12.1.4 Transfer Error Status Register** for more information.

If the bus monitor is enabled, then $\overline{\text{TEA}}$ is asserted. The $\overline{\text{TEA}}$ signal will generate a machine check exception if it occurs between a $\overline{\text{TS}}$ and $\overline{\text{TA}}$ signal. When a $\overline{\text{TA}}$ is asserted before a write-protection error is detected, the $\overline{\text{TEA}}$ that occurs will not generate a machine check exception. Refer to **Section 12.4 The Bus Monitor** for more information.

Note: If the bus monitor is disabled and the write-protect error occurs, $\overline{\text{TEA}}$ assertion will not occur. See **Section 12.12.1.4 Transfer Error Status Register** for more information.

Bits 9–15—Reserved

These bits are reserved and should be set to 0.

15.3.1.4 MEMORY COMMAND REGISTER. The memory command register (MCR) allows you to issue commands to stimulate UPM routine execution. This capability enables the CPU to perform special memory operations in addition to the standard read/write and periodic timer service operations.

MCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	OP		RESERVED						UM	RESERVED						
RESET	0		0						0	0						
R/W	R/W		R/W						R/W	R/W						
ADDR	(IMMR & 0xFFFF0000) + 0x168															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	MB			RES	MCLF				RESERVED		MAD					
RESET	0			0	0				0		0					
R/W	R/W			R/W	R/W				R/W		R/W					
ADDR	(IMMR & 0xFFFF0000) + 0x16A															

OP—Command Opcode

This field defines the operation to be executed by the user-programmable machine that is specified in the UM field.

00 = Writes the contents of the memory data register into the RAM location indexed by the MAD field.
01 = Reads the contents of the RAM location indexed by the MAD field and stores it in the memory data register.
10 = Executes the RAM word in the RAM array that services one of the eight memory banks specified in the MB field. The executed RAM word is referenced by the MAD field. If the executed RAM word has the LAST bit set, it will be the last RAM word executed.
11 = Reserved.

Bits 2–7—Reserved

These bits are reserved and should be set to 0.

UM—User Machine

This bit selects the user-programmable machine for this command.

0 = UPMA.
1 = UPMB.

Bits 9–15—Reserved

These bits are reserved and should be set to 0.

MB—Memory Bank

This field selects the appropriate $\overline{CSx}$ pin when a **RUN** command is executed.

000 = $\overline{CS0}$ is selected.
001 = $\overline{CS1}$ is selected.
010 = $\overline{CS2}$ is selected.
011 = $\overline{CS3}$ is selected.
100 = $\overline{CS4}$ is selected.
101 = $\overline{CS5}$ is selected.
110 = $\overline{CS6}$ is selected.
111 = $\overline{CS7}$ is selected.

Bits 19 and 24–25—Reserved

These bits are reserved and should be set to 0.

MCLF—Memory Command Loop Field

This field specifies the number of times a loop is executed for any command.

0001 = The loop is executed 1 time.
0010 = The loop is executed 2 times.
0011 = The loop is executed 3 times.
0100 = The loop is executed 4 times.
0101 = The loop is executed 5 times.
0110 = The loop is executed 6 times.
0111 = The loop is executed 7 times.
1000 = The loop is executed 8 times.
1001 = The loop is executed 9 times.
1010 = The loop is executed 10 times.
1011 = The loop is executed 11 times.
1100 = The loop is executed 12 times.
1101 = The loop is executed 13 times.
1110 = The loop is executed 14 times.
1111 = The loop is executed 15 times.
0000 = The loop is executed 16 times.

MAD—Memory Array Index

This field specifies an index to one of 64 RAM words in the RAM array for command execution.

15.3.1.5 MACHINE A MODE REGISTER. The machine A mode register (MAMR) contains the configuration for the user-programmable machine A. See Figure 15-1 (page 15-3) for more information.

MAMR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PTA								PTAE	AMA			RES	DSA		RES
RESET	0								0	0			0	0		0
R/W	R/W								R/W	R/W			R/W	R/W		R/W
ADDR	(IMMR & 0xFFFF0000) + 0x170															

BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	GOCLA			GPLA4 DIS	RLFA				WLFA				TLFA			
RESET	0			1	0				0				0			
R/W	R/W			R/W	R/W				R/W				R/W			
ADDR	(IMMR & 0xFFFF0000) + 0x172															

PTA—Periodic Timer A Period

This field affects the periodic timer A and determines the timer period service rate according to the following equation:

$$PTA = \frac{\text{System Clock (MHz)} \times \text{Service Duration } (\mu s)}{2^{2 \times DFBRG} \times \text{Prescaler (PTP)} \times NCS}$$

NCS is an integer between 1 and 8 that represents the number of enabled chip-selects that select this UPM. The DFBRG field is the division factor for the BRGCLK, which can be divide by 1 (default), 4, 16, or 64 and is programmed in the system clock and reset control register (SCCR) in **Section 5.2 Register Model**.

For example, for DRAM to maintain data integrity an access or refresh must occur every 15.6µs. Use the equation above to determine the PTA value for UPMA to perform memory refresh. Given that you have a 25MHz system clock with the required service rate of 15.6µs, a periodic timer prescaler equal to 32, and a DFBRG field that is equal to 0, then the PTA value should be $(25 \times 15.6) / (2^{2\times0} \times 32 \times 1) = 12$. If you want to perform more than one refresh per service, use the TFLA field.

PTAE—Periodic Timer A Enable

This bit allows the periodic timer A to request service.

0 = Periodic timer A is disabled.
1 = Periodic timer A is enabled.

AMA—Address Multiplex Size A

This field specifies the number of address lines to be output on the bus at the first clock of the memory cycle. Refer to Table 15-7 (page 15-60) for more information. The AMX field of the RAM array entry controls the output of the address lines. Refer to Table 15-3 (page 15-38) for more information. For example, these address lines can be used to connect to the DRAM devices that require row and columns to be multiplexed on the same pin.

Bits 12 and 15—Reserved

These bits are reserved and should be set to 0.

DSA—Disable Timer Period

This bit guarantees a minimum time between accesses to the same memory bank if it is controlled by the UPMA. The TODT bit turns on the disable timer in the RAM array and, when expired, the UPMA allows the machine access to issue a memory pattern to the same memory region. Accesses to different memory regions using two or more chip-selects can be handled by this same UPMA, assuming they have the same timing. The maximum disable period is four clock cycles. When switching to a different bank that requires more than four clock cycles, you must add more UPM RAM word to meet your time requirement. Refer to **Section 15.5.4.2 RAM Word Operation** for more specific DRAM example information.

00 = 1-cycle disable period.
01 = 2-cycle disable period.
10 = 3-cycle disable period.
11 = 4-cycle disable period.

G0CLA—General Line 0 Control A

This field selects the address line that is output to the internal $\overline{\text{GPL0}}$ signal when the UPMA is selected to control memory access.

000 = A12.
001 = A11.
010 = A10.
011 = A9.
100 = A8.
101 = A7.
110 = A6.
111 = A5.

GPLA4DIS—GPLA4 Output Line Disable

This bit determines whether or not the UPWAITA/$\overline{\text{GPL\_A4}}$ pin will behave as an output line controlled by the internal $\overline{\text{GPL4}}$ signal and the UPM RAM word.

0 = UPWAITA/$\overline{\text{GPL\_A4}}$ is defined as $\overline{\text{GPL\_A4}}$.
1 = UPWAITA/$\overline{\text{GPL\_A4}}$ is defined as UPWAITA.

RLFA—Read Loop Field A

This field specifies the number of times a loop defined in the UPMA RAM word is executed for a burst read or single beat read cycle.

0001 = The loop is executed 1 time.
0010 = The loop is executed 2 times.
0011 = The loop is executed 3 times.
0100 = The loop is executed 4 times.
0101 = The loop is executed 5 times.
0110 = The loop is executed 6 times.
0111 = The loop is executed 7 times.
1000 = The loop is executed 8 times.
1001 = The loop is executed 9 times.
1010 = The loop is executed 10 times.
1011 = The loop is executed 11 times.
1100 = The loop is executed 12 times.
1101 = The loop is executed 13 times.
1110 = The loop is executed 14 times.
1111 = The loop is executed 15 times.
0000 = The loop is executed 16 times.

WLFA—Write Loop Field A

This field specifies the number of times a loop defined in the UPMA RAM word is executed for a burst write or a single beat write cycle.

0001 = The loop is executed 1 time.
0010 = The loop is executed 2 times.
0011 = The loop is executed 3 times.
0100 = The loop is executed 4 times.
0101 = The loop is executed 5 times.
0110 = The loop is executed 6 times.
0111 = The loop is executed 7 times.
1000 = The loop is executed 8 times.
1001 = The loop is executed 9 times.
1010 = The loop is executed 10 times.
1011 = The loop is executed 11 times.
1100 = The loop is executed 12 times.
1101 = The loop is executed 13 times.
1110 = The loop is executed 14 times.
1111 = The loop is executed 15 times.
0000 = The loop is executed 16 times.

TLFA—Timer Loop Field A

This field specifies the number of times a loop defined in the UPMA RAM word is executed for a periodic timer service.

0001 = The loop is executed 1 time.
0010 = The loop is executed 2 times.
0011 = The loop is executed 3 times.
0100 = The loop is executed 4 times.
0101 = The loop is executed 5 times.
0110 = The loop is executed 6 times.
0111 = The loop is executed 7 times.
1000 = The loop is executed 8 times.
1001 = The loop is executed 9 times.
1010 = The loop is executed 10 times.
1011 = The loop is executed 11 times.
1100 = The loop is executed 12 times.
1101 = The loop is executed 13 times.
1110 = The loop is executed 14 times.
1111 = The loop is executed 15 times.
0000 = The loop is executed 16 times.

15.3.1.6 MACHINE B MODE REGISTER. The machine B mode register (MBMR) contains the configuration for the user-programmable B machine. See Figure 15-1 (page 15-3) for more information.

MBMR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PTB								PTBE	AMB			RES	DSB		RES
RESET	0								0	0			0	0		0
R/W	R/W								R/W	R/W			R/W	R/W		R/W
ADDR	(IMMR & 0xFFFF0000) + 0x174															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	G0CLB			GPLB4 DIS	RLFB				WLFB				TLFB			
RESET	0			1	0				0				0			
R/W	R/W			R/W	R/W				R/W				R/W			
ADDR	(IMMR & 0xFFFF0000) + 0x176															

PTB—Periodic Timer B

This field affects the periodic timer B and determines the timer period according to the following equation:

$$PTB = \frac{\text{System Clock (MHz)} \times \text{Service Duration } (\mu s)}{2^{2 \times DFBRG} \times \text{Prescaler (PTP)} \times NCS}$$

NCS is an integer between 1 and 8 that represents the number of enabled chip-selects that select this UPM. The DFBRG field is the division factor for the BRGCLK, which can be divide by 1 (default), 4, 16, or 64 and is programmed in the system clock and reset control register (SCCR) in **Section 5.2 Register Model**.

For example, for DRAM to maintain data integrity an access or refresh must occur every 15.6µs. Use the equation above to determine the PTB value for UPMB to perform memory refresh. Given that you have a 25MHz system clock with the required service rate of 15.6µs, a periodic timer prescaler equal to 32, and a DFBRG field that is equal to 0, then the PTB value should be $(25 \times 15.6) / (2^{2 \times 0} \times 32 \times 1) = 12$. If you want to perform more than one refresh per service, use the TFLB field.

PTBE—Periodic Timer B Enable

This bit allows the periodic timer B to request service.

0 = Periodic timer B is disabled.
1 = Periodic timer B is enabled.

AMB—Address Multiplex Size B

This field specifies the number of address lines to be output on the bus at the first clock of the memory cycle. Refer to Table 15-7 (page 15-60) for more information. The AMX field of the RAM array entry controls the output of the address lines, as shown in Table 15-3 (page 15-38). For example, these address lines can be used to connect to the DRAM devices that require row and columns to be multiplexed on the same pin.

Bits 12 and 15—Reserved

These bits are reserved and should be set to 0.

DSB—Disable Timer Period

This bit guarantees a minimum time between accesses to the same memory bank if it is controlled by the UPMB. The TODT bit turns on the disable timer in the RAM array and, when expired, the UPMB allows the machine access to issue a memory pattern to the same memory region. Accesses to different memory regions using two or more chip-selects can be handled by this same UPMB, assuming they have the same timing. The maximum disable period is four clock cycles. When switching to a different bank that requires more than four clock cycles, you must add more UPM RAM word to meet your time requirement. Refer to **Section 15.5.4.2 RAM Word Operation** for more specific DRAM example information.

00 = 1-cycle disable period.
01 = 2-cycle disable period.
10 = 3-cycle disable period.
11 = 4-cycle disable period.

G0CLB—General Line 0 Control B

This field selects the address line that is output to the internal $\overline{\text{GPL0}}$ signal when the UPMB is selected to control memory access.

000 = A12.
001 = A11.
010 = A10.
011 = A9.
100 = A8.
101 = A7.
110 = A6.
111 = A5.

GPLB4DIS—GPLB4 Output Line Disable

This bit determines whether or not the UPWAITB/$\overline{\text{GPL\_B4}}$ pin will behave as an output line controlled by the internal $\overline{\text{GPL4}}$ signal and the UPM RAM word.

0 = UPWAITB/$\overline{\text{GPL\_B4}}$ is defined as $\overline{\text{GPL\_B4}}$.
1 = UPWAITB/$\overline{\text{GPL\_B4}}$ is defined as UPWAITB.

RLFB—Read Loop Field B

This field specifies the number of times a loop defined in the UPMB RAM word is executed for a burst read or single beat read cycle.

0001 = The loop is executed 1 time.
0010 = The loop is executed 2 times.
0011 = The loop is executed 3 times.
0100 = The loop is executed 4 times.
0101 = The loop is executed 5 times.
0110 = The loop is executed 6 times.
0111 = The loop is executed 7 times.
1000 = The loop is executed 8 times.
1001 = The loop is executed 9 times.
1010 = The loop is executed 10 times.
1011 = The loop is executed 11 times.
1100 = The loop is executed 12 times.
1101 = The loop is executed 13 times.
1110 = The loop is executed 14 times.
1111 = The loop is executed 15 times.
0000 = The loop is executed 16 times.

WLFB—Write Loop Field B

This field specifies the number of times a loop defined in the UPMB RAM word is executed for a burst write or a single beat write cycle.

0001 = The loop is executed 1 time.
0010 = The loop is executed 2 times.
0011 = The loop is executed 3 times.
0100 = The loop is executed 4 times.
0101 = The loop is executed 5 times.
0110 = The loop is executed 6 times.
0111 = The loop is executed 7 times.
1000 = The loop is executed 8 times.
1001 = The loop is executed 9 times.
1010 = The loop is executed 10 times.
1011 = The loop is executed 11 times.
1100 = The loop is executed 12 times.
1101 = The loop is executed 13 times.
1110 = The loop is executed 14 times.
1111 = The loop is executed 15 times.
0000 = The loop is executed 16 times.

TLFB—Timer Loop Field B

This field specifies the number of times a loop defined in the UPMB RAM word is executed for a periodic timer service cycle.

0001 = The loop is executed 1 time.
0010 = The loop is executed 2 times.
0011 = The loop is executed 3 times.
0100 = The loop is executed 4 times.
0101 = The loop is executed 5 times.
0110 = The loop is executed 6 times.
0111 = The loop is executed 7 times.
1000 = The loop is executed 8 times.
1001 = The loop is executed 9 times.
1010 = The loop is executed 10 times.
1011 = The loop is executed 11 times.
1100 = The loop is executed 12 times.
1101 = The loop is executed 13 times.
1110 = The loop is executed 14 times.
1111 = The loop is executed 15 times.
0000 = The loop is executed 16 times.

15.3.1.7 MEMORY DATA REGISTER. The memory data register (MDR) contains the data to be written to or read from the RAM array for UPM command operations. This register must be set up before you issue a write command to the memory command register.

MDR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	MD															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x17C															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	MD															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x17E															

MD—Memory Data

This field contains the RAM array word.

15.3.1.8 MEMORY ADDRESS REGISTER. The memory address register (MAR) contains an address to be output on the address lines that are controlled by the AMX field in the RAM word of the RAM array.

MAR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	MA															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x164															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	MA															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x166															

MA—Memory Address

This field contains a 32-bit address to be output on the address bus if the AMX field is equal to 11. Refer to **Section 15.5.4 The RAM Array** for more information.

15.3.1.9 MEMORY PERIODIC TIMER PRESCALER REGISTER. The memory periodic timer prescaler register (MPTPR) defines the divisor of the BRGCLK used as the memory periodic timer input clock. Refer to **Section 5.3.4 Internal Clock Signals** for details.

MPTPR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PTP								RESERVED							
RESET	00000001								00000000							
R/W	R/W								R/W							
ADDR	(IMMR & 0xFFFF0000) + 0x17A															

PTP—Periodic Timers Prescaler
This field determines the division factor that is shown below.

001x xxxx = Divide by 2.
0001 xxxx = Divide by 4.
0000 1xxx = Divide by 8.
0000 01xx = Divide by 16.
0000 001x = Divide by 32.
0000 0001 = Divide by 64.
1xxx xxxx = Reserved.
01xx xxxx = Reserved.

Bits 8–15—Reserved
These bits are reserved and should be set to 0.

15.4 THE GENERAL-PURPOSE CHIP-SELECT MACHINE

The general-purpose chip-select machine (GPCM) allows a glueless and flexible interface between the MPC823, SRAM, EPROM, FEPROM, ROM devices, and external peripherals. The GPCM contains three basic register groups that you can use to configure it—base registers 0–7, option registers 0–7, and the memory status register.

15.4.1 Configuration

If the MS field in the BRx of the selected bank selects the general-purpose chip-select machine, the attributes for the memory cycle initiated are taken from the ORx. These attributes include the CSNT, ACS, SCY, TRLX, EHTR, and SETA fields. See Table 15-2 for signal behavior and system response.

Anywhere from 0 to 30 wait states can be programmed for $\overline{TA}$ generation. The $\overline{WEx}$ signals are available for each byte that is written to memory. Also, an $\overline{OE}$ signal is provided to eliminate external glue logic. On system reset, a global chip-select ($\overline{CS0}$) is asserted to provide a boot ROM chip-select before the system is fully configured.

Table 15-2. GPCM Strobe Signal Behavior

OPTION REGISTER ATTRIBUTES					SIGNAL BEHAVIOR						
TRLX	ACCESS TYPE	EBDF	CSNT	ACS	ADDRESS TO $\overline{CSx}$ ASSERTED	ADDRESS TO $\overline{OE}$ ASSERTED	ADDRESS TO $\overline{WEx}$ ASSERTED	DATA TO $\overline{WEx}$ ASSERTED	$\overline{CSx}$ NEGATED TO ADD/DATA INVALID	$\overline{WEx}$ NEGATED TO ADD/DATA INVALID	TOTAL NUMBER OF CYCLES
0	Read	—	—	00	0	3/4*Clock	—	—	1/4* Clock	—	2+SCY
				10	1/4*Clock						
				11	1/2*Clock						
	Write	—	0	00	0	—	3/4*Clock	-1/4*Clock		1/4*Clock	
				10	1/4*Clock						
				11	1/2*Clock						
		00	1	00	0				1/2*Clock	1/2*Clock	
				10	1/4*Clock						
				11	1/2*Clock						
		01		00	0				1/4*Clock	3/8*Clock	
				10	1/4*Clock				3/8*Clock		
				11	1/2*Clock						
1	Read	—	—	00	0	3/4*Clock	—	—	1/4*Clock	—	2+2*SCY
				10	(1+1/4)*Clock	1+3/4*Clock					3+2*SCY
				11	(1+1/2)*Clock	1+3/4*Clock					
	Write	—	0	00	0	—	3/4*Clock	-1/4*Clock		1/4*Clock	2+2*SCY
				10	(1+1/4)*Clock		1+3/4*Clock	3/4*Clock			3+2*SCY
				11	(1+1/2)*Clock		1+3/4*Clock	3/4*Clock			
		00	1	00	0		3/4*Clock	-1/4*Clock		1+1/2*Clock	
				10	(1+1/4)*Clock		1+3/4*Clock	3/4*Clock	1+1/2*Clock		4+2*SCY
				11	(1+1/2)*Clock		1+3/4*Clock	3/4*Clock			
		01		00	0		3/4*Clock	-1/4*Clock	1/4*Clock	3/8*Clock	3+2*SCY
				10	(1+1/4)*Clock		1+3/4*Clock	3/4*Clock	3/8*Clock		4+2*SCY
				11	(1+1/2)*Clock		1+3/4*Clock	3/4*Clock			

NOTE: SCY is the number of wait cycles from the option register.

Next, the banks selected by the general-purpose chip-select machine support an option to output the $\overline{CSx}$ signal at different timings with respect to the external address bus. $\overline{CSx}$ can be output in any of the following configurations:

- Simultaneous with the external address
- One quarter of a clock later
- One half of a clock later

This all depends, of course, on the value of the ACS field, plus an additional cycle if the TRLX bit is set. The general-purpose chip-select machine allows you to connect to devices that have long disconnect times on data by delaying new bus transactions addressing other memory banks for additional clock cycles. Finally, the banks selected to operate with the general-purpose chip-select machine support termination of an external cycle by sensing the $\overline{TA}$ signal asserted by the addressed external slave. Refer to Table 15-2 for more information.

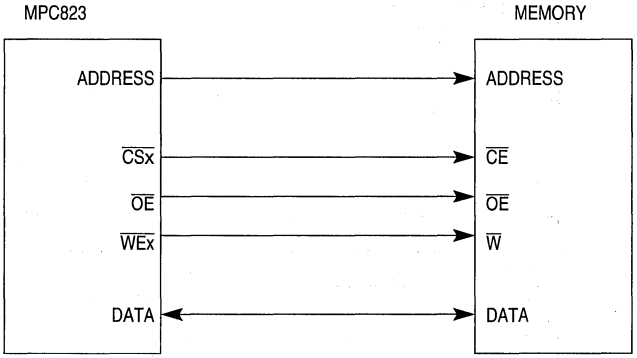


Figure 15-5. GPCM Memory Device Interface

Figure 15-5 illustrates a basic connection between the MPC823 and a "static" memory device. In this case, the $\overline{CSx}$ signal is connected directly to the $\overline{CE}$ signal of the memory device. The $\overline{WEx}$ signals are connected to the respective $\overline{W}$ signal in the memory device where each $\overline{WEx}$ signal corresponds to a different data byte.

As illustrated in Figure 15-6, the timing of the $\overline{CSx}$ signal is the same as the timing of the address lines. The strobes for the transaction are supplied by the $\overline{OE}$ or $\overline{WEx}$ signals, depending on the transaction direction (read or write). The negation of the $\overline{WEx}$ signal is controlled by the CSNT bit. The $\overline{CSx}$ signal is generated when the ACS field in the corresponding ORx register is set to '00'.

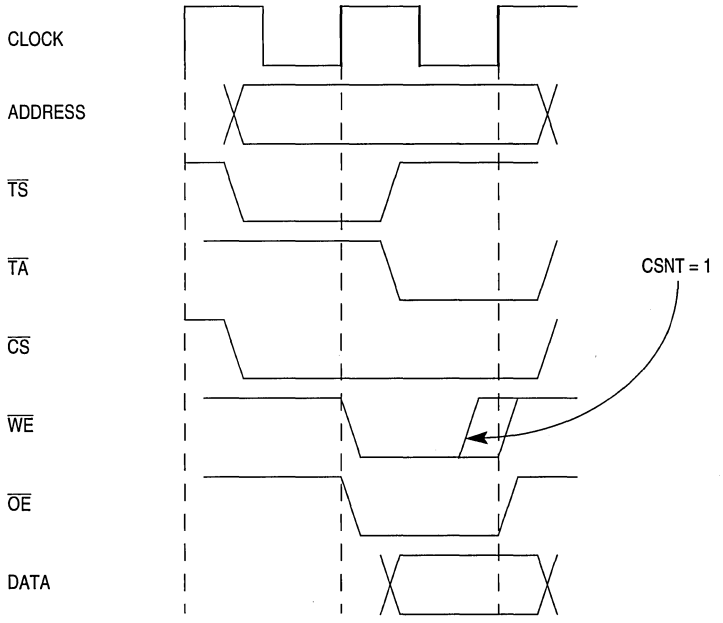


Figure 15-6. GPCM Memory Device Basic Timing (ACS = 00, CSNT = 1, and TRLX = 0)

Figure 15-7 illustrates the basic connection between the MPC823 and an external peripheral device. In this case, $\overline{CSx}$ is connected directly to the $\overline{CE}$ signal of the memory device and the R/$\overline{W}$ signal is connected to the respective R/$\overline{W}$ signal in the peripheral device. The $\overline{CSx}$ signal is the strobe output for the memory access.

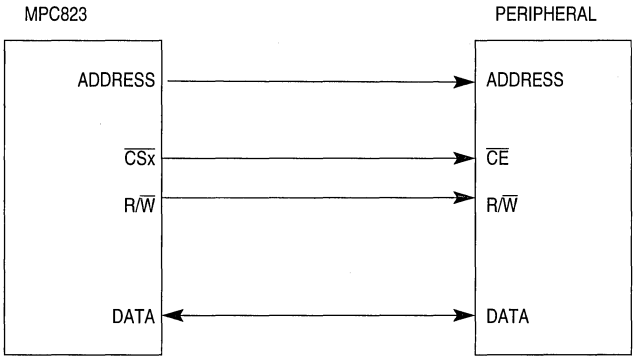


Figure 15-7. GPCM Peripheral Device Interface

Figure 15-8 illustrates the $\overline{CSx}$ signal as defined by the setup time required between the address lines and the $\overline{CE}$ signal. The MPC823 memory controller allows you to specify the $\overline{CSx}$ signal to meet this requirement using the ACS field in the option register.

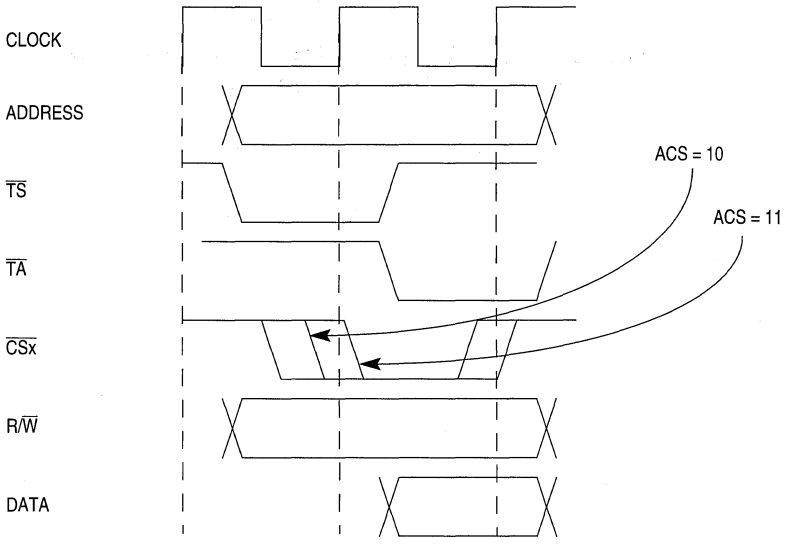


Figure 15-8. GPCM Peripheral Device Basic Timing (ACS = 10 or 11 and TRLX = 0)

The general-purpose chip-select machine also provides a CSNT attribute in the option register that controls the timing for the appropriate strobe negation in write cycles. When this attribute is asserted, the strobe is negated one quarter of a clock before the normal case. For example, when the ACS field equals 00 and CSNT is set, $\overline{WEx}$ is negated one quarter of a clock earlier and when ACS does not equal 00 and CSNT is set, $\overline{WEx}$ and $\overline{CSx}$ are negated one quarter of a clock earlier. For more information, see Figure 15-6, Figure 15-8, and Table 15-2.

The TRLX field in the option register is provided for memory systems that require more relaxed timing between signals. When TRLX is set and the ACS field is not equal to 00, an additional cycle between the address and strobes is inserted by the MPC823 memory controller, as shown in Figure 15-9.

When TRLX and CSNT are set in a write-memory access, the strobe lines ($\overline{WEx}$ and $\overline{CSx}$, if ACS is not equal to 00) are negated one clock earlier than in the normal case, as shown in Figure 15-11. When a bank is selected to operate with external transfer acknowledge (SETA and TRLX are set), the memory controller does not support external devices that provide the $\overline{TA}$ signal to complete the transfer with zero wait states. The minimum access duration in this case is three clock cycles.

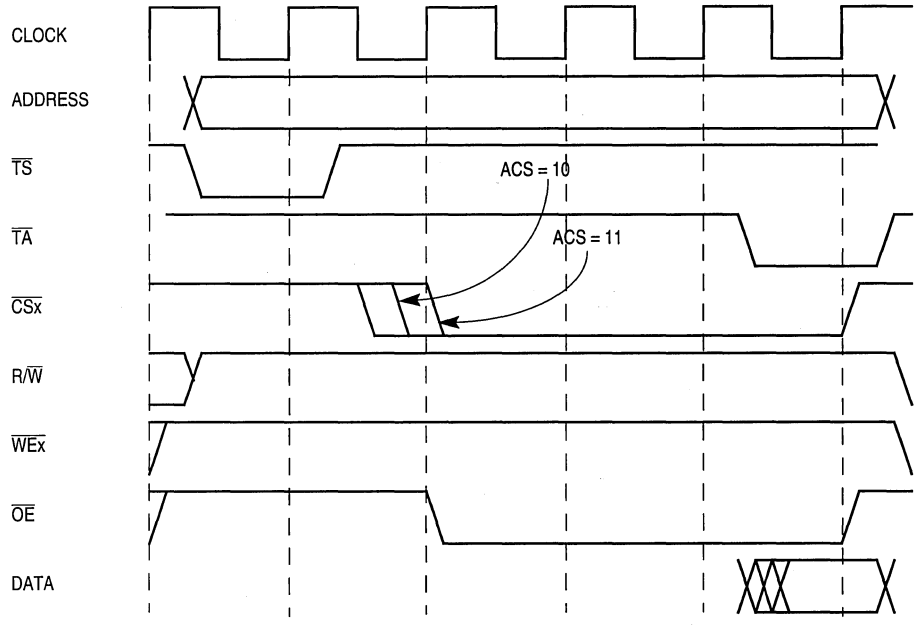


Figure 15-9. MPC823 GPCM–Relaxed Timing–Read Access (ACS = 10 or 11, SCY = 1, and TRLX =1)

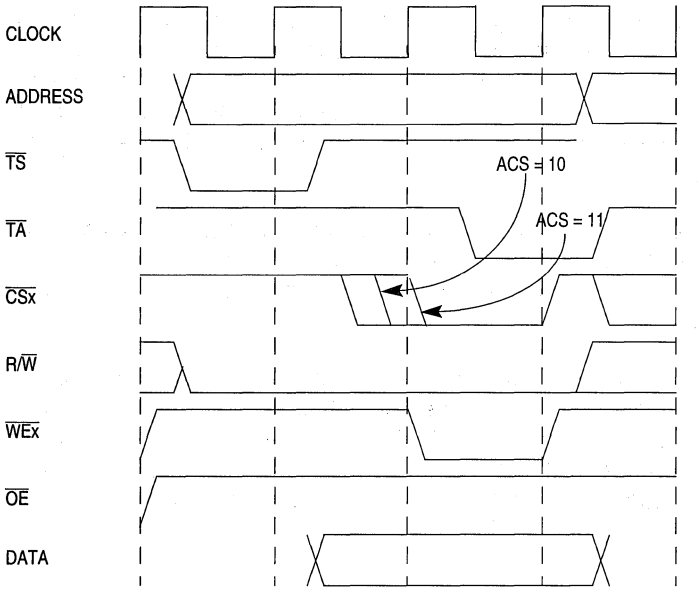


Figure 15-10. MPC823 GPCM–Relaxed Timing–Write Access (ACS = 10 or 11, SCY = 0, CSNT = 0, and TRLX =1)

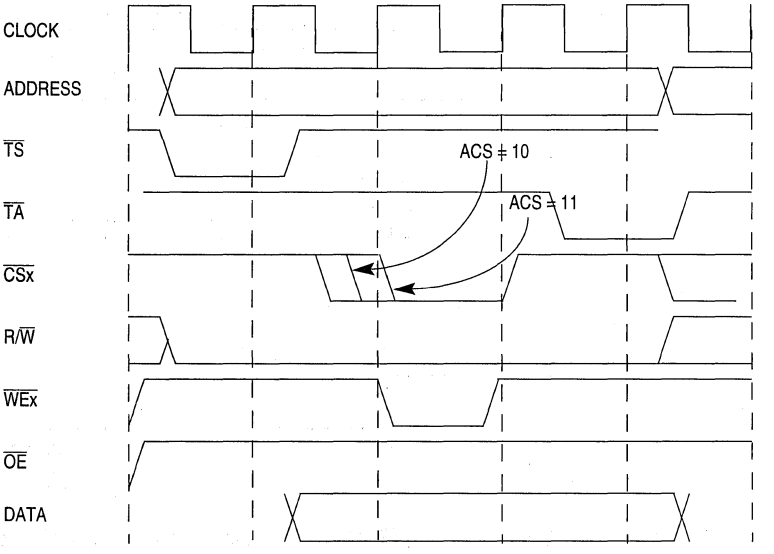


Figure 15-11. MPC823 GPCM–Relaxed Timing–Write Access (ACS = 10 or 11, SCY = 0, CSNT = 1, and TRLX =1)

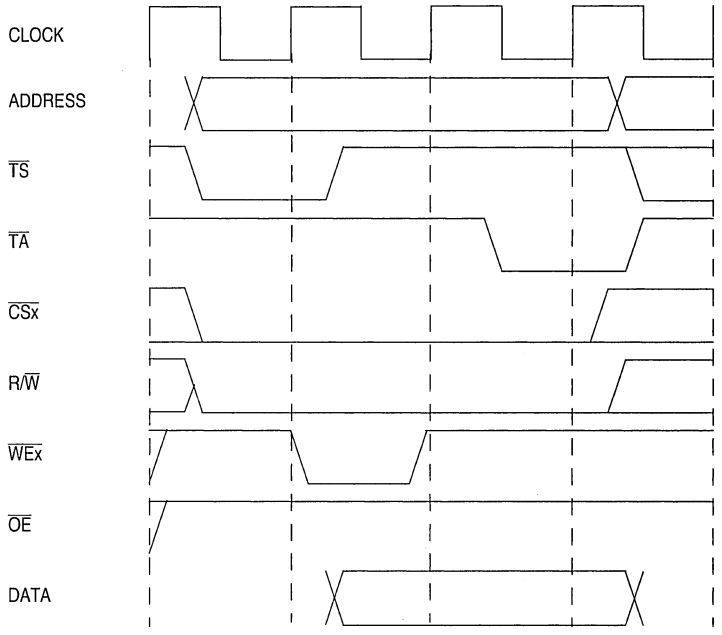


Figure 15-12. MPC823 GPCM–Relaxed Timing–Write Access (ACS = 00, SCY = 0, CSNT = 1, and TRLX =1)

15.4.1.1 PROGRAMMABLE WAIT STATE CONFIGURATION. The general-purpose chip-select machine supports internal TA signal generation. It allows "fast" accesses to external memory through an internal bus master or it allows a maximum 17-clock access. This can be done by programming the SCY field in the option register. The internal TA generation mode is enabled if the SETA field in the option register is cleared. If the TA pin is externally asserted at least two clock cycles before the wait state counter has expired, the current memory cycle is terminated. When the TRLX bit is set, the number of wait states inserted by the memory controller is defined by 2 x SCY or a maximum of 30 wait states.

15.4.1.2 EXTENDED HOLD TIME ON READ ACCESSES. Slow memory devices that require a long delay on data read accesses should set the EHTR field in the corresponding option register. Any GPCM access to the external bus following a read access to the slower memory bank is delayed by one clock cycle, unless it is a read access to the same bank. Refer to Figure 15-13 through Figure 15-16 for details.

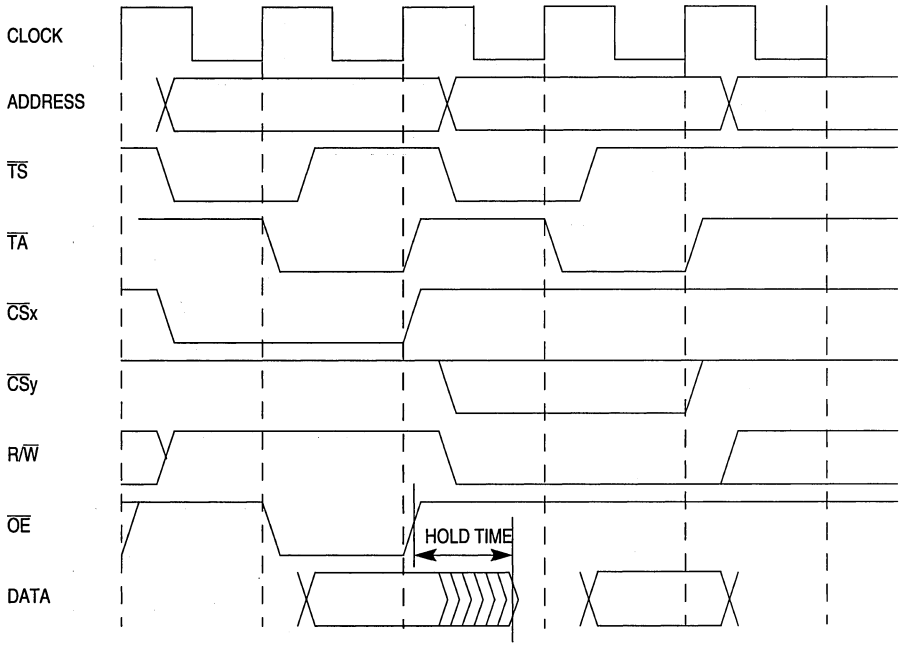


Figure 15-13. GPCM Read Followed By Write (EHTR = 0)

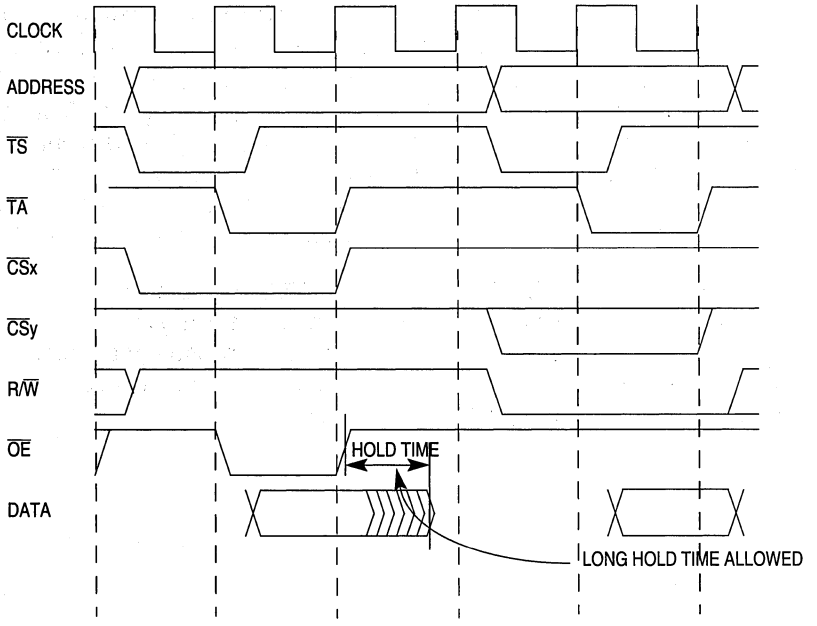


Figure 15-14. GPCM Write Followed By Read (EHTR = 1)

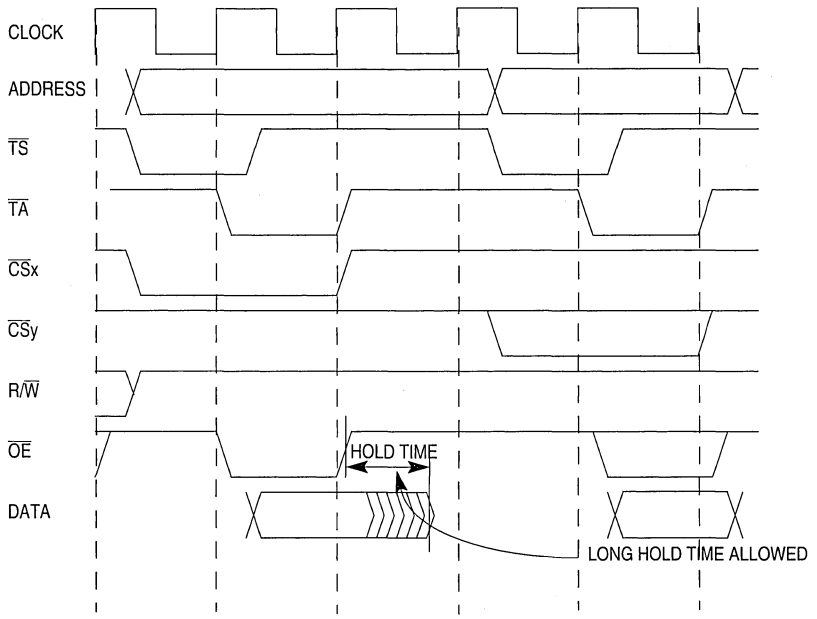


Figure 15-15. GPCM Read Followed By Read From Different Banks (EHTR = 1)

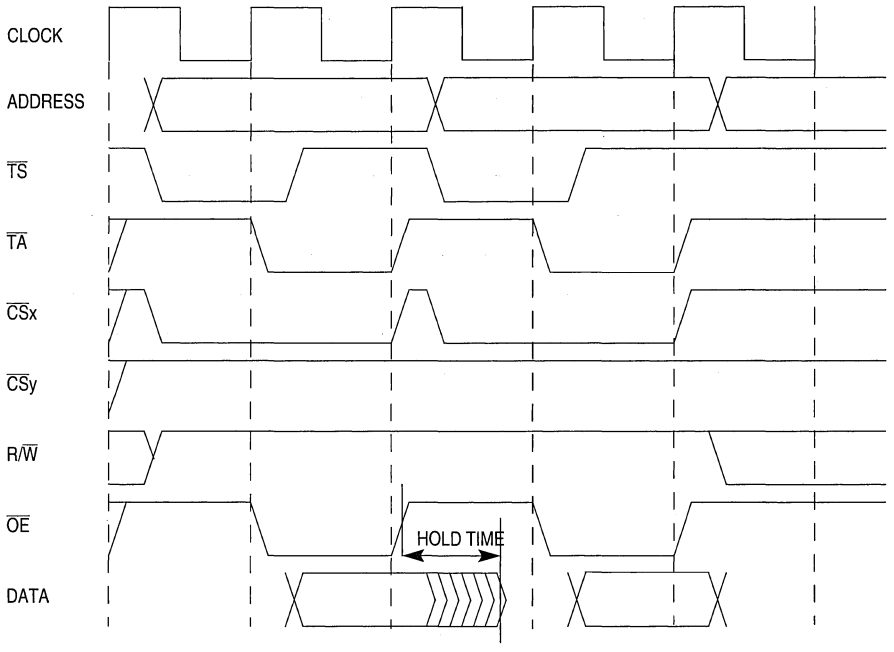


Figure 15-16. GPCM Read Followed By Read From Same Bank (EHTR = 1)

15.4.1.3 BOOT CHIP-SELECT OPERATION. Boot chip-select operation allows address decoding for a boot ROM before system initialization occurs. The $\overline{CS0}$ signal is the boot chip-select output and its operation differs from the other external chip-select outputs on system reset. When the MPC823 internal core begins accessing memory at system reset, $\overline{CS0}$ is asserted for every address, unless an internal register is accessed.

The boot chip-select provides a programmable port size during system reset by using the BPS field of the hard reset configuration word, as shown in **Section 4.3.1.1 Hard Reset Configuration Word**. Setting these appropriately allows a boot ROM to be located anywhere in the address space. The boot chip-select does not provide write protection and responds to all address types. $\overline{CS0}$ operates this way until the first write to the option register 0 and it can be used as any other chip-select register once the preferred address range is loaded into base register 0. After the first write to option register 0, the boot chip-select can only be restarted on system reset. The initial values of the "boot bank" in the memory controller are described in Table 15-3.

Table 15-3. Boot Bank Field Values After Reset

	FIELD	VALUE
Base Register 0	PS	From Hard Reset Configuration Word
	PARE	0
	WP	0
	MS	00
	V	From Hard Reset Configuration Word
Option Register 0	AM	00000000000000000
	ATM	000
	CSNT	1
	ACS	11
	$\overline{BI}$	1
	SCY	1111
	SETA	0
	TRLX	1
	EHTR	0

15.4.1.4 SRAM INTERFACE. Figure 15-17 illustrates a simple connection between an SRAM device and the MPC823.

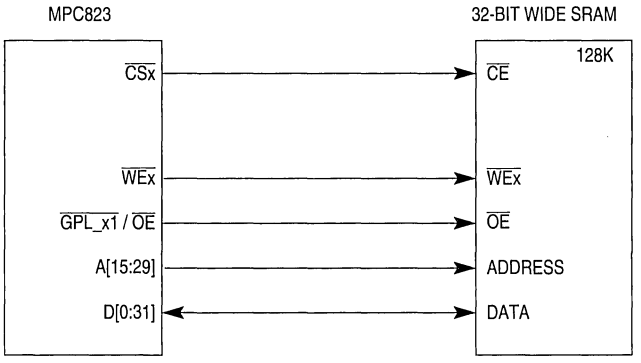


Figure 15-17. GPCM to SRAM Configuration

15.4.1.5 EXTERNAL ASYNCHRONOUS MASTER SUPPORT. Figure 15-18 illustrates the basic interface between an asynchronous external master and the GPCM to allow connection to "static RAM" memory.

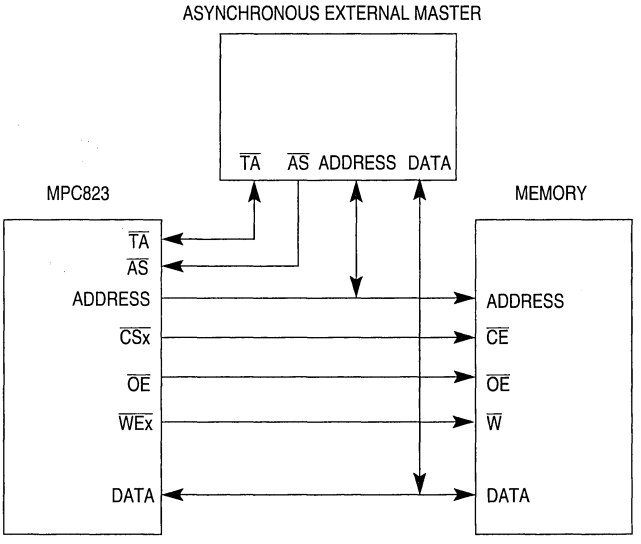


Figure 15-18. Asynchronous External Master Configuration For GPCM-Handled Memory Devices

Figure 15-19 illustrates the timing for TRLX = 0 when an external asynchronous master accesses SRAM. The $\overline{TA}$ signal remains asserted with the $\overline{WE}$ and $\overline{OE}$ signals until the $\overline{AS}$ signal is negated by the external master.

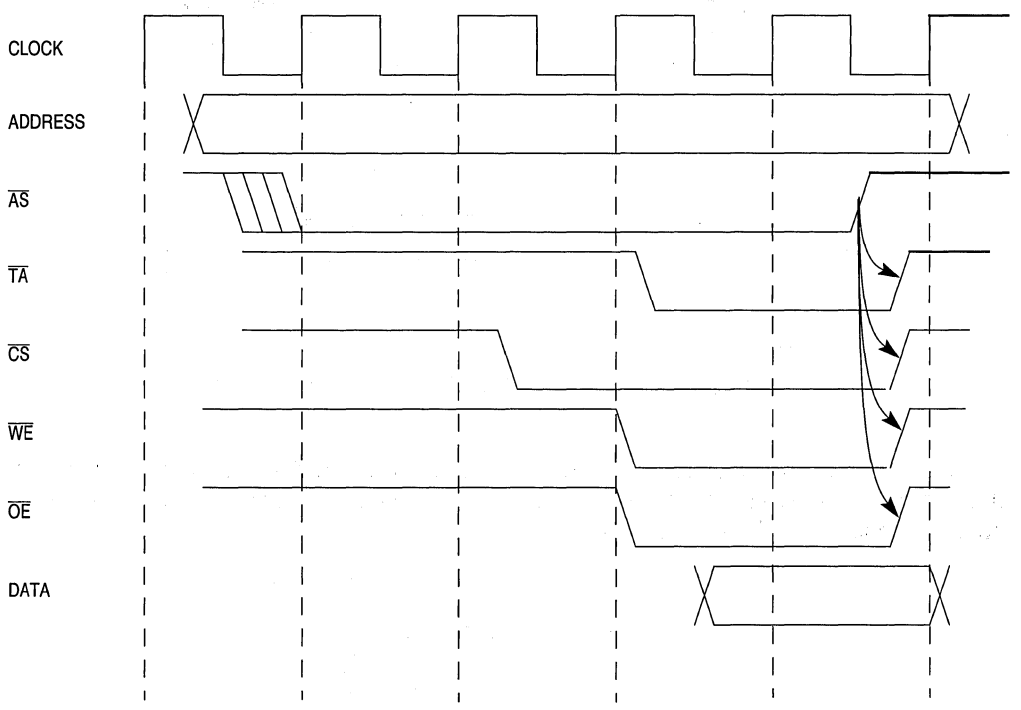


Figure 15-19. Asynchronous External Master, GPCM-Handled Memory Access Timing (TRLX = 0)

When an external asynchronous master performs an access to a memory device via the general-purpose chip-select machine in the memory controller, the CSNT bit in the option register is configured as "don't care".

15.5 USER-PROGRAMMABLE MACHINES

Each of the two user-programmable machines (UPMs) is a flexible interface that connects to a wide range of memory devices. At the heart of each UPM is an internal memory RAM array that specifies what the logical value driven on the external memory controller pins are for a given clock cycle. Each word in the RAM array provides bits that allow a memory access to be controlled with a resolution of one quarter of the system clock period on the byte-select and chip-select lines. Figure 15-20 illustrates the basic operation of each UPM. The following basic methods can be used to initiate a UPM cycle:

- Any internal or external master requests an external memory access
- A memory periodic timer expires and requests a transaction
- A transfer error or reset generates an exception request
- The memory command register receives a **RUN** command (software) from the CPU

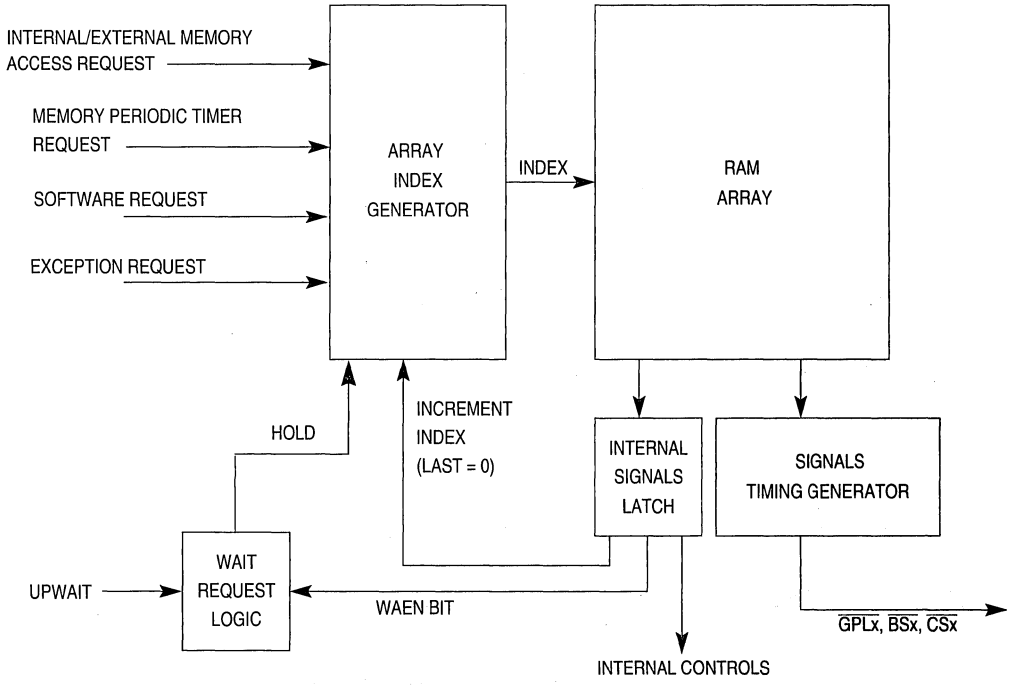


Figure 15-20. User-Programmable Machine Block Diagram

The RAM array contains 32-bit entries referred to as RAM words. If the UPM reads a RAM word with the WAEN bit set, the external UPWAITx signal is sampled and synchronized by the memory controller and the current request is frozen. The signal timing generator will load the RAM word from the RAM array to drive the general-purpose lines, byte-selects, and chip-selects.

15.5.1 Requests

The user-programmable machine has four basic requests that can initiate a UPM cycle. There is a special start address in the RAM array that is associated with each of the following cycle types:

- Read single beat start address (RSS)
- Read burst cycle start address (RBS)
- Write single beat start address (WSS)
- Write burst cycle start address (WBS)
- Periodic timer start address (PTS)
- Exception condition start address (EXS)

Figure 15-21 illustrates the first locations addressed by the UPM, according to the different cycle types. Software requests, however, can point to any of the 64 UPM RAM entries.

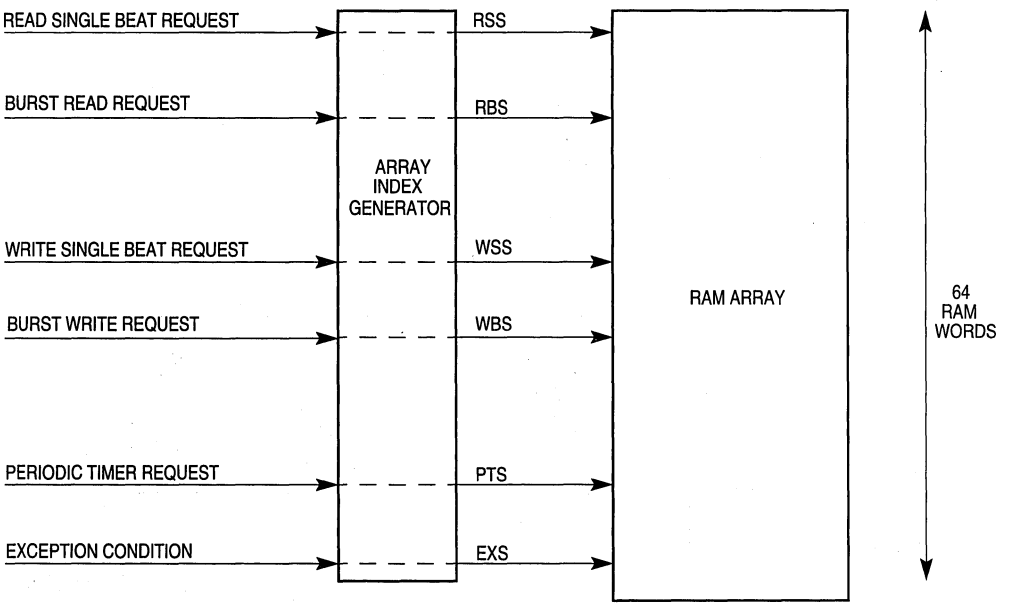


Figure 15-21. RAM Array Indexing

15.5.1.1 INTERNAL/EXTERNAL MEMORY ACCESS REQUESTS. When any of the internal masters request a new access to external memory, the address and type of the transfer are compared to each one of the valid banks defined in the base register. The value of the MS field in the base register selects the UPM that will handle the memory access. You must ensure that the appropriate UPM entries are created prior to your request.

The external memory access requests consist of read single beat, read burst, write single beat, and write burst. A single beat cycle is generated by the master to a cache-inhibited memory bank. A typical burst cycle is generated to the memory that allows multiple accesses. It only occurs when your memory is burstable. A single beat cycle starts out with one transfer start and ends with one transfer acknowledge. For a 32-bit access, the burst cycle starts out with one transfer start but ends with four transfer acknowledges. For a 16-bit bus, there are eight transfer acknowledges. For an 8-bit bus, there are 16 transfer acknowledges.

15.5.1.2 MEMORY PERIODIC TIMER REQUESTS. Each UPM contains a periodic timer that can be programmed to generate periodic service requests that will be indexed into the RAM array. Figure 15-22 illustrates the hardware associated with memory periodic timer request generation. In general, the periodic timer is used for refresh cycle operation.

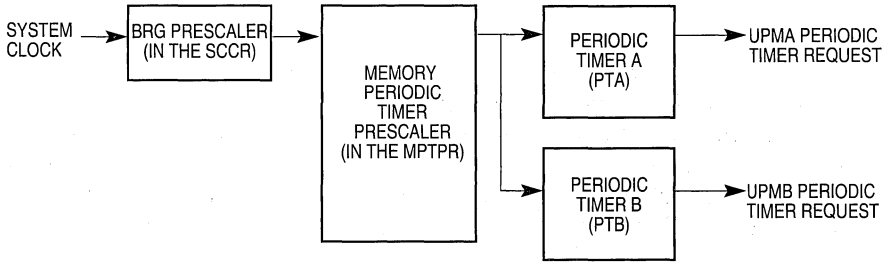


Figure 15-22. Memory Periodic Timer Request Block Diagram

15.5.1.3 SOFTWARE REQUESTS. The software can initiate a request to the user-programmable machine by issuing one of three commands—read, write, or execute a RAM word— to the memory command register. Every memory device has its own signal handshaking protocol to put it into self-refresh mode or any special protocol mode. In the user-programmable machine there are unused areas that enable you to write a special RAM word for this protocol. Any unused area in the UPM RAM can be used to store these RAM words. Typically, software requests are used to put the memory in self-refresh mode. You can use this method to maintain memory integrity before going into low-power or lower power modes. A new command must be issued to exit self-refresh mode or any special protocol mode after returning to normal operation.

15.5.1.4 EXCEPTION REQUESTS. When an access to a memory device is initiated by the MPC823 under UPM control, the external device may assert the $\overline{\text{TEA}}$, $\overline{\text{SRESET}}$, or $\overline{\text{HRESET}}$ signal. The UPM provides a mechanism that allows you to handle the memory control signals to meet the timing requirements of the device and assume no data is lost.

15.5.2 Programming the User-Programmable Machine

The user-programmable machine is a micro-sequencer that requires micro-instructions or RAM words to generate signal timings for different memory cycles. You should program the user-programmable machine in the following order:

1. Write a program into the RAM array.
2. Set up the base and option registers.
3. Program the memory periodic timer prescaler register.
4. Program the machine mode register.

Each user-programmable machine has a machine mode register (MxMR) that defines the general attributes for operation. The PTA field of the MAMR and the PTB field of the MBMR defines the period for the timers associated with UPMA and UPMB. If the PTAE bit is set, the periodic timer of UPMA requests a transaction when the timer period expires. If the PTBE bit is set, the periodic timer of UPMB requests a transaction when the timer period expires.

To initiate a software request, issue the appropriate command to the memory command register with the MAD field indexing the first entry of the UPM entry word. Command execution is accomplished by accessing consecutive RAM words (one per clock) until the word with the LAST bit set is encountered. The words read from the RAM provide information about the value and timing of the external signals controlled by the UPM and about specific strobes that control internal memory controller resources.

There is a disable timer mechanism associated with each user-programmable machine that is only active between memory accesses. This timer is used to provide a delay between successive memory cycles to the same bank.

Each of the UPMs can control how the address of the current access is output to the external pins. Address multiplexing configurations for a specific memory or device can be selected in the machine mode register. There is also a multiplexing field in the RAM word that is used to control cycle-by-cycle accesses. Specific user-programmable machine register information is located in **Section 15.3.1 Register Descriptions**.

15.5.3 Clock Timing

The RAM word includes fields that specify the value of the various external signals at each clock edge. The signal timing generator causes the external signals to behave according to the timing specified in the current RAM word. Figure 15-23 and Figure 15-24 illustrate the clock schemes of the user-programmable machines in the memory controller. The clock phases shown in these figures reflect timing windows that specify when generated signals can change state. Figure 15-23 represents the clock scheme selected when the EBDF field of the system clock and reset control register (SCCR) is equal to 00. As indicated in the figure, CLKOUT is the same as system clock. In Figure 15-24, if the EBDF field of the system clock and reset control register is equal to 01, then CLKOUT is equal to the system clock divided by 2. Notice that in this scheme GCLK1 does not have a 50% duty cycle. The state of the external signals may change (if specified in the RAM array) at any edge of GCLK1 and GCLK2, plus a propagation delay.

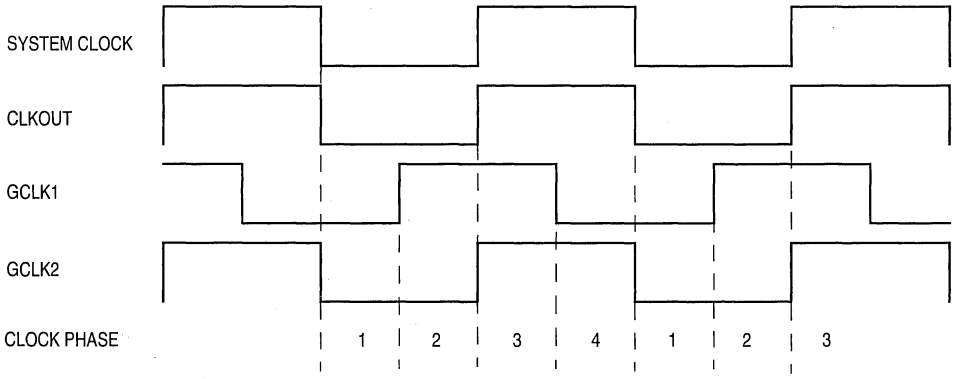


Figure 15-23. UPM Clock Scheme One (Division Factor = 1)

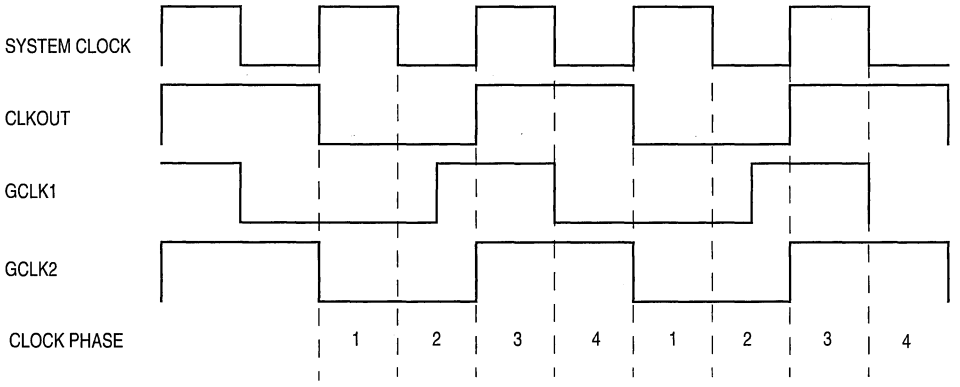


Figure 15-24. UPM Clock Scheme Two (Division Factor = 2

The $\overline{CSx}$ signals are handled in a similar way, except that only the $\overline{CSx}$ signal corresponding to the currently accessed bank is modified. The $\overline{BS}$ signal assertion and negation timing is also specified for each cycle in the RAM word, but the final value of each one of these signals depends on the port size of the specified bank, the external address accessed, and the value of the TSIZx pins.

Figure 15-25 and Figure 15-26 provide examples of how to control the timing of the $\overline{CSx}$, $\overline{GPL1}$, and $\overline{GPL2}$ signals A RAM word is read on the rising edge of every GCLK2 cycle or in phase 3 of the previous clock cycle. It determines the value of the CST1–4, G1T3, G1T4, G2T3, and G2T4 bits, which specifies the timing of chip-selects, byte-selects, and GPLx signals based on any edge of GCLK1 or GCLK2. The clock phases shown in these figures refer to the timing windows when the signals controlled by these bits in the RAM word are driven.

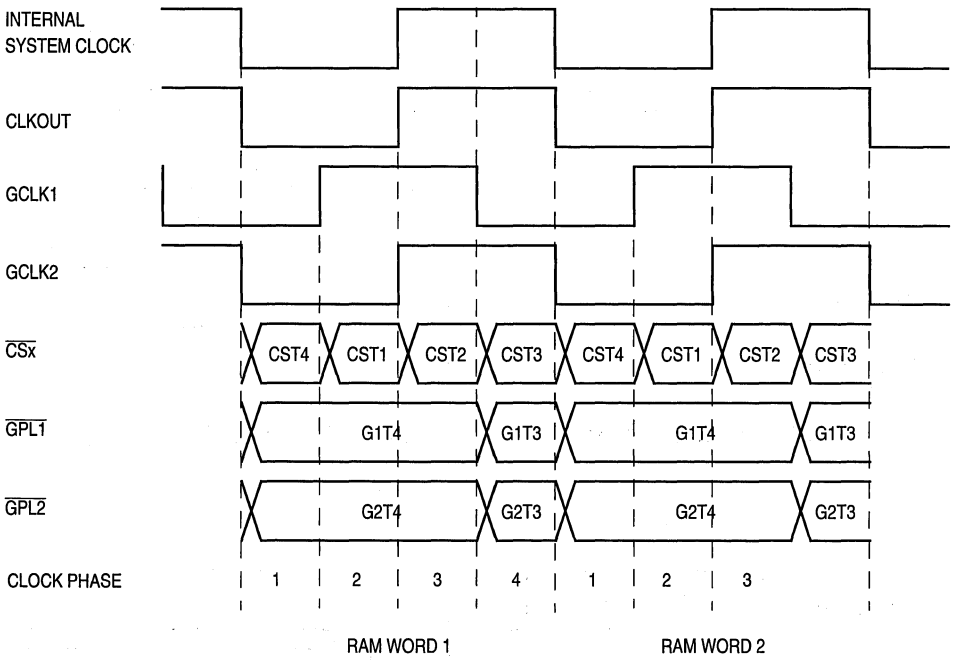


Figure 15-25. UPM Signals Timing Example One (Division Factor = 1, EBDF = 00)

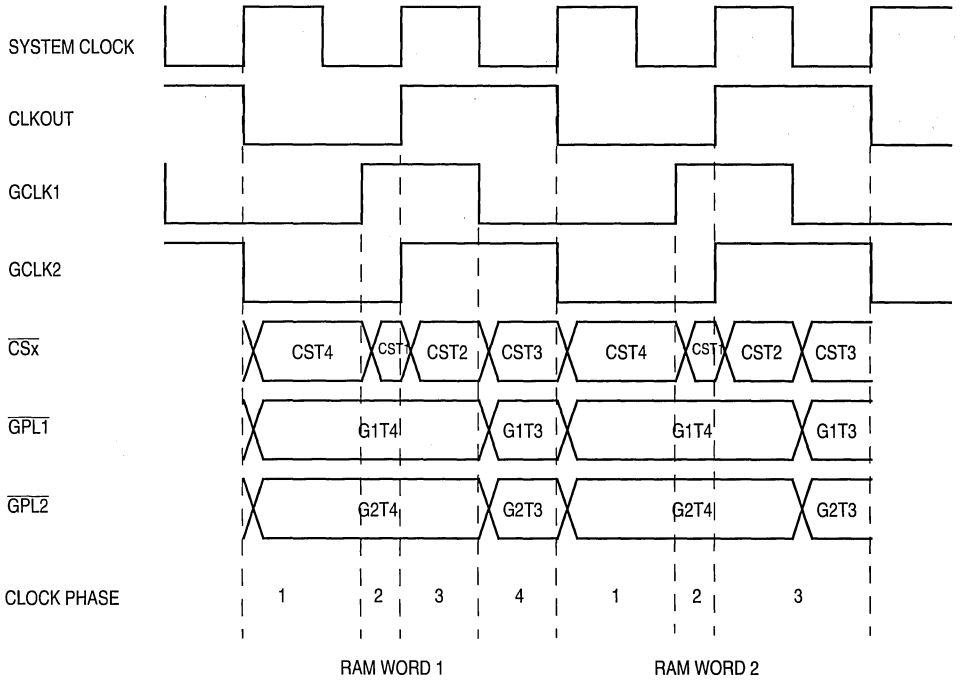


Figure 15-26. UPM Signals Timing Example Two (Division Factor = 2, EBDF = 01)

15.5.4 The RAM Array

The RAM array size for each UPM is 64 locations deep and 32 bits wide as illustrated in Figure 15-27. The selected bank is one of eight banks that matches the current address. The signal generation shown at the bottom part of the figure are outputs of the UPM and not direct signal outputs of the MPC823.

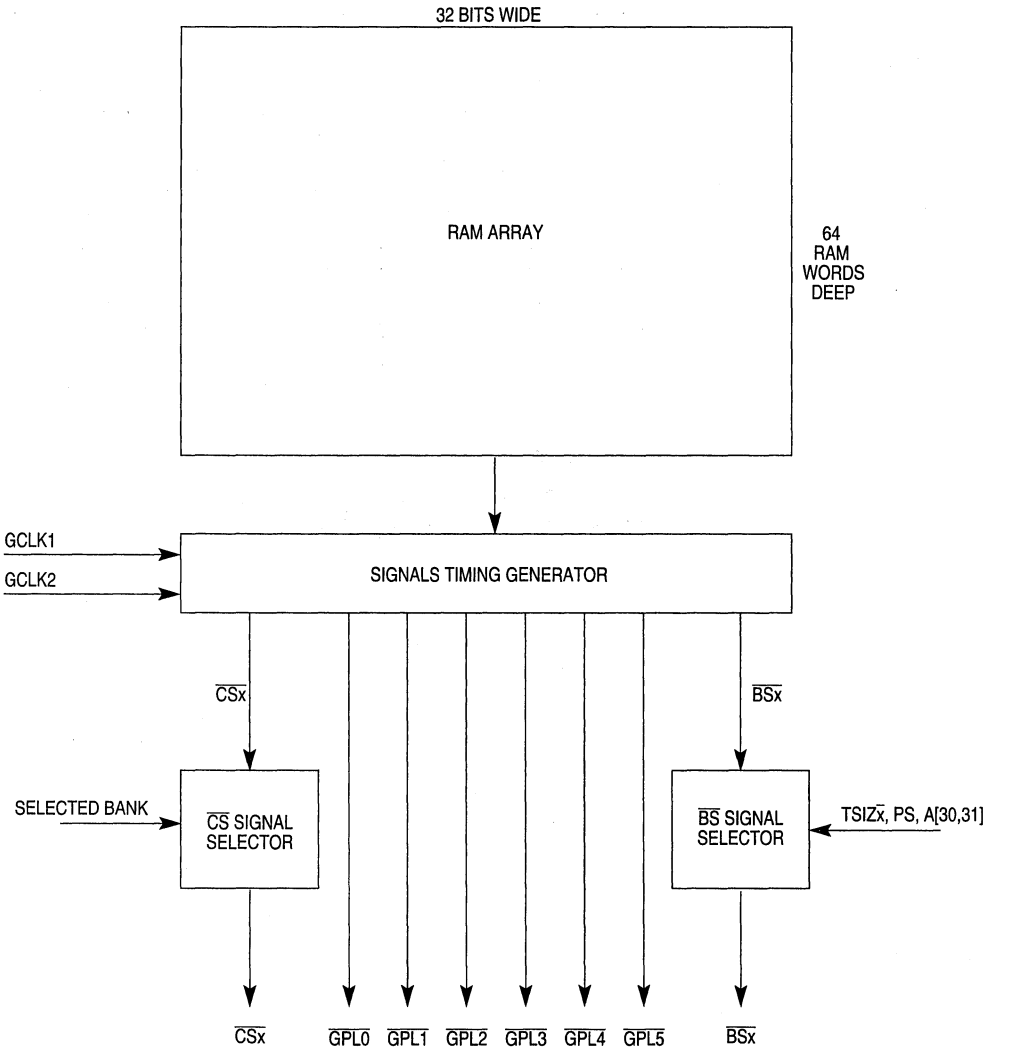


Figure 15-27. RAM Array and Signal Generation

15.5.4.1 THE RAM WORD. The RAM word is a 32-bit wide micro-instruction that is stored in one of 64 locations in the RAM array.

15.5.4.1.1 RAM Word Format. The RAM word format selects and specifies the timing of all external signals controlled by the user-programmable machine.

RAM WORD

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	CST4	CST1	CST2	CST3	BST4	BST1	BST2	BST3	G0L		G0H		G1T4	G1T3	G2T4	G2T3
RESET	—	—	—	—	—	—	—	—	—		—		—	—	—	—
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W		R/W	R/W	R/W	R/W
ADDR	(MCR_{MAD}) INDIRECT ADDRESSING OF 1 OF 64 ENTRIES															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	G3T4	G3T3	G4T4/ DLT3	G4T3/ WAEN	G5T4	G5T3	RESERVED		LOOP	EXEN	AMX		NA	UTA	TODT	LAST
RESET	—	—	—	—	—	—	—		—	—	—		—	—	—	—
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W		R/W	R/W	R/W	R/W
ADDR	*															

NOTE: * All 32 bits of the RAM word are addressed as shown in the address row above.
— = Undefined.

CST4—Chip-Select Timing 4

This bit defines the state of the $\overline{CSx}$ signal during clock phase 1.

0 = The $\overline{CSx}$ signal is asserted at the trailing edge of GCLK2.
1 = The $\overline{CSx}$ signal is negated at the trailing edge of GCLK2.

Note: The state of the selected $\overline{CSx}$ signal depends on the value of each CSTx bit and the corresponding bank.

CST1—Chip-Select Timing 1

This bit defines the state of the $\overline{CSx}$ signal during clock phase 2.

0 = The $\overline{CSx}$ signal is asserted at the rising edge of GCLK1.
1 = The $\overline{CSx}$ signal is negated at the rising edge of GCLK1.

CST2—Chip-Select Timing 2

This bit defines the state of the $\overline{CSx}$ signal during clock phase 3.

0 = The $\overline{CSx}$ signal is asserted at the rising edge of GCLK2.
1 = The $\overline{CSx}$ signal is negated at the rising edge of GCLK2.

CST3—Chip-Select Timing 3

This bit defines the state of the $\overline{CSx}$ signal during clock phase 4.

0 = The $\overline{CSx}$ signal is asserted at the trailing edge of GCLK1.
1 = The $\overline{CSx}$ signal is negated at the trailing edge of GCLK1.

BST4—Byte-Select Timing 4

This bit defines the state of the $\overline{BSx}$ signal during clock phase 1.

0 = The $\overline{BSx}$ signal is asserted at the trailing edge of GCLK2.
1 = The $\overline{BSx}$ signal is negated at the trailing edge of GCLK2.

Note: The state of each $\overline{BSx}$ signal depends on the value of each BSTx bit and three other parameter values—the PS field in the selected base register and the TSIZx and A[30:31] signals in the currently accessed cycle.

BST1—Byte-Select Timing 1

This bit defines the state of the $\overline{BSx}$ signal during clock phase 2.

0 = The $\overline{BSx}$ signal is asserted at the rising edge of GCLK1.
1 = The $\overline{BSx}$ signal is negated at the rising edge of GCLK1.

BST2—Byte-Select Timing 2

This bit defines the state of the $\overline{BSx}$ signal during clock phase 3.

0 = The $\overline{BSx}$ signal is asserted at the rising edge of GCLK2.
1 = The $\overline{BSx}$ signal is negated at the rising edge of GCLK2

BST3—Byte-Select Timing 3

This bit defines the state of the $\overline{BSx}$ signal during clock phase 4.

0 = The $\overline{BSx}$ signal is asserted at the trailing edge of GCLK1.
1 = The $\overline{BSx}$ signal is negated at the trailing edge of GCLK1.

G0L—General-Purpose Line 0 Lower

This field defines the state of the $\overline{GPL0}$ signal during phases 1 through 3.

10 = The $\overline{GPL0}$ signal is asserted at the trailing edge of GCLK2.
11 = The $\overline{GPL0}$ signal is negated at the trailing edge of GCLK2.
00 = The $\overline{GPL0}$ signal is driven at the trailing edge of GCLK2 as defined in the G0CLx field of the MxMR.

G0H—General-Purpose Line 0 Higher

This field defines the state of the $\overline{GPL0}$ signal during phase 4.

10 = The $\overline{GPL0}$ signal is asserted at the trailing edge of GCLK1.
11 = The $\overline{GPL0}$ signal is negated at the trailing edge of GCLK1.
00 = The $\overline{GPL0}$ signal is driven at the trailing edge of GCLK1 as defined in the G0CLx field of the MxMR.

G1T4—General-Purpose Line 1 Timing 4

This bit defines the state of the $\overline{GPL1}$ signal during phases 1 through 3.

0 = The $\overline{GPL1}$ signal is asserted at the trailing edge of GCLK2.
1 = The $\overline{GPL1}$ signal is negated at the trailing edge of GCLK2.

G1T3—General-Purpose Line 1 Timing 3

This bit defines the state of the $\overline{GPL1}$ signal during phase 4.

0 = The $\overline{GPL1}$ signal is asserted at the trailing edge of GCLK1.
1 = The $\overline{GPL1}$ signal is negated at the trailing edge of GCLK1.

G2T4— General-Purpose Line 2 Timing 4

This bit defines the state of the $\overline{GPL2}$ signal during phases 1 through 3.

0 = The $\overline{GPL2}$ signal is asserted at the trailing edge of GCLK2.
1 = The $\overline{GPL2}$ signal is negated at the trailing edge of GCLK2.

G2T3—General-Purpose Line 2 Timing 3

This bit defines the state of the $\overline{GPL2}$ signal during phase 4

0 = The $\overline{GPL2}$ signal is asserted at the trailing edge of GCLK1.
1 = The $\overline{GPL2}$ signal is negated at the trailing edge of GCLK1.

G3T4—General-Purpose Line 3 Timing 4

This bit defines the state of the $\overline{GPL3}$ signal during phases 1 through 3.

0 = The $\overline{GPL3}$ signal is asserted at the trailing edge of GCLK2.
1 = The $\overline{GPL3}$ signal is negated at the trailing edge of GCLK2.

G3T3—General-Purpose Line 3 Timing 3

This bit defines the state of the $\overline{GPL3}$ signal during phase 4.

0 = The $\overline{GPL3}$ signal is asserted at the trailing edge of GCLK1.
1 = The $\overline{GPL3}$ signal is negated at the trailing edge of GCLK1.

G4T4/DLT3—General-Purpose Line 4 Timing 4/Delay Time 3

This bit performs two functions depending on the value of the GPLx4DIS bit in the machine mode register. If the MxMR defines the UPWAITx/GPL_x4 pin as an output (GPL_x4), then this bit functions as G4T4. If it is defined as an input (UPWAITx), then this bit functions as DLT3.

If you have configured GPLx4DIS = 0 in the MxMR, then you have selected G4T4:

0 = The value of the GPL4 signal at the trailing edge of GCLK2 will be 0.
1 = The value of the GPL4 signal at the trailing edge of GCLK2 will be 1.

If you have configured GPLx4DIS = 1 in the MxMR, then you have selected UPWAITx and DLT3 is the controlling function:

0 = The data bus should be sampled at the rising edge of GCLK2 for all reads.
1 = The data bus should be sampled at the falling edge of GCLK2 for all reads.

G4T3/WAEN—General-Purpose Line 4 Timing 3/Wait Enable

This bit performs two functions depending on the value of the GPLx4DIS bit in the machine mode register. If the MxMR defines the UPWAITx/GPL_x4 pin as an output (GPL_x4), then this bit functions as G4T3. If it is defined as an input (UPWAITx), then this bit functions as WAEN.

If you have configured GPLx4DIS = 0 in the MxMR, then you have selected G4T3:

0 = The value of the GPL4 signal at the trailing edge of GCLK1 will be 0.
1 = The value of the GPL4 signal at the trailing edge of GCLK1 will be 1.

If you have configured GPLx4DIS = 1 in the MxMR, then you have selected UPWAITx and WAEN is the controlling function:

0 = The UPWAITx function is disabled.
1 = A "freeze" in the logical value of the UPM-controlled external signals will occur when the UPWAITx pin is asserted. The UPWAITx signal is sampled on the trailing edge of GCLK2. See Figure 15-33 for more information.

G5T4—General-Purpose Line 5 Timing 4

This bit defines the state of the GPL5 signal during phases 1 through 3.

0 = The value of the GPL5 signal at the trailing edge of GCLK2 will be 0.
1 = The value of the GPL5 signal at the trailing edge of GCLK2 will be 1.

G5T3—General-Purpose Line 5 Timing 3

This bit defines the state of the GPL5 signal during phase 4.

0 = The value of the GPL5 signal at the trailing edge of GCLK1 will be 0.
1 = The value of the GPL5 signal at the trailing edge of GCLK1 will be 1.

Bits 22 and 23—Reserved

These bits are reserved and should be set to 0.

LOOP—Loop

The first RAM word in the RAM array where LOOP is 1 is recognized as the Loop Start Word. The next RAM word where LOOP is 1 is recognized as the Loop End Word. The RAM words between the start and end are defined as the loop. The UPM executes this loop as many times as it is defined in the corresponding loop field of the MxMR.

0 = The current RAM word is not the start or end of a loop construct.
1 = The current RAM word is the start or end of a loop construct.

EXEN—Exception Enable

When an external device asserts the $\overline{\text{TEA}}$ or $\overline{\text{RESET}}$ signals, this bit allows you to branch to the exception start address (EXS) where you would store your exception handler. The exception start address is found at a fixed address in the RAM array.

0 = The UPM continues executing the remaining RAM words.
1 = The current RAM word allows a branch to an exception handler after the current cycle if an exception condition is detected. The exception condition can be an external device asserting $\overline{\text{TEA}}$, $\overline{\text{HRESET}}$, or $\overline{\text{SRESET}}$.

AMX—Address Multiplexing

This bit determines the source of the A[0:31] signals.

00 = The value of the A[6:31] signals at the trailing edge of GCLK1 is the address that is requested by the internal master. For example, column address.
01 = Reserved.
10 = The value of the A[6:31] signals at the trailing edge of GCLK1 is the address that is requested by the internal master multiplexed according to the AMA/AMB field of the MxMR. For example, row address.
11 = The value of the A[6:31] signals at the trailing edge of GCLK1is the contents of the memory address register (MAR). For example, SDRAM mode initialization.

NA—Next Address

This bit determines how much the current address is incremented.

0 = The address increment is disabled
1 = In conjunction with the PS field in the base register, the increment value of the A[28:31] signals at the trailing edge of GCLK1 is as follows:
— If the accessed bank has a 32-bit port size, the value of the A[28:31] signals are incremented by 4.
— If the accessed bank has a 16-bit port size, the value of the A[28:31] signals are incremented by 2.
— If the accessed bank has an 8-bit port size, the value of the A[28:31] signals are incremented by 1.

Note: The value of the NA bit is only relevant when the UPM serves a burst-read or burst-write request. Under other patterns this bit is reserved.

UTA—UPM Transfer Acknowledge

This bit controls the state of the $\overline{TA}$ signal sampled by the external bus interface in the current memory cycle. The $\overline{TA}$ signal is output at the rising edge of GCLK2.

0 = The $\overline{TA}$ signal is driven low on the next rising edge of GCLK2.
1 = The $\overline{TA}$ signal is driven high on the next rising edge of GCLK2.

TODT—Turn On Disable Timer

This bit controls the disable timer mechanism.

0 = The disable timer is turned off.
1 = The disable timer for the currently accessed bank is activated. This prevents a new access to the same bank (when controlled by the UPMs) until the disable timer expires. For example, precharge time.

LAST—Last

If this bit is set, it is the last RAM word in the program.

0 = The UPM continues executing RAM words.
1 = The service to the UPM request is done.

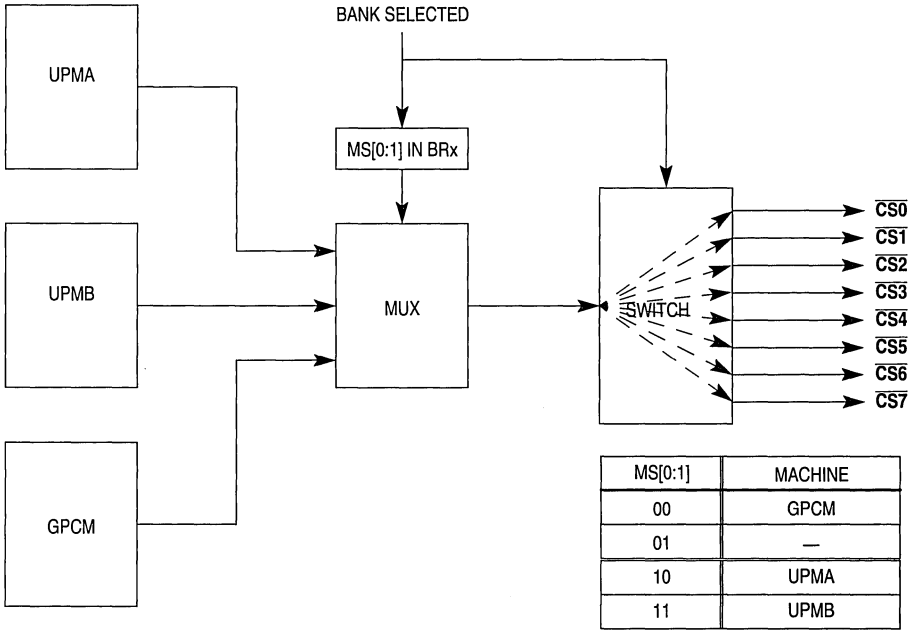
15.5.4.2 RAM WORD OPERATION. This section describes how the RAM word affects the behavior of the chip-select, byte-select, general-purpose, transfer acknowledgment signals, as well as address multiplexing and the wait mechanism.

15.5.4.2.1 Start Addresses. Each UPM request has a special address, except for software requests, which can start at any RAM word. Table 15-4 provides the start addresses of the UPM RAM words for each request type.

Table 15-4. Start Address Locations

REQUEST TO BE SERVICED	UPM START ADDRESS
Read Single Beat Cycle (RSS)	0x'00
Read Burst Cycle (RBS)	0x'08
Write Single Beat Cycle (WSS)	0x'18
Write Burst Cycle (WBS)	0x'20
Periodic Timer Request (PTS)	0x'30
Exception (EXS)	0x'3C

15.5.4.2.2 Chip-Select Signals. The MS field in the base register of the accessed memory bank selects a user-programmable machine on the currently requested cycle. The selected UPM only affects the assertion and negation of the appropriate $\overline{CSx}$ signal and its timing is specified in the UPM RAM word. Figure 15-28 illustrates how the $\overline{CSx}$ signals are controlled by the UPMs.



MS[0:1]	MACHINE
00	GPCM
01	—
10	UPMA
11	UPMB

Figure 15-28. $\overline{CSx}$ Signal Selection

15.5.4.2.3 Byte-Select Signals. The MS field in the base register of the accessed memory bank selects a user-programmable machine on the currently requested cycle. The selected UPM only affects the assertion and negation of the appropriate $\overline{BSx}$ signal and its timing as specified in the RAM word. The $\overline{BSx}$ signals are also controlled by the port size of the accessed bank, the transfer size of the transaction, and the address accessed. Figure 15-29 illustrates how the $\overline{BSx}$ signals are controlled by the user-programmable machines.

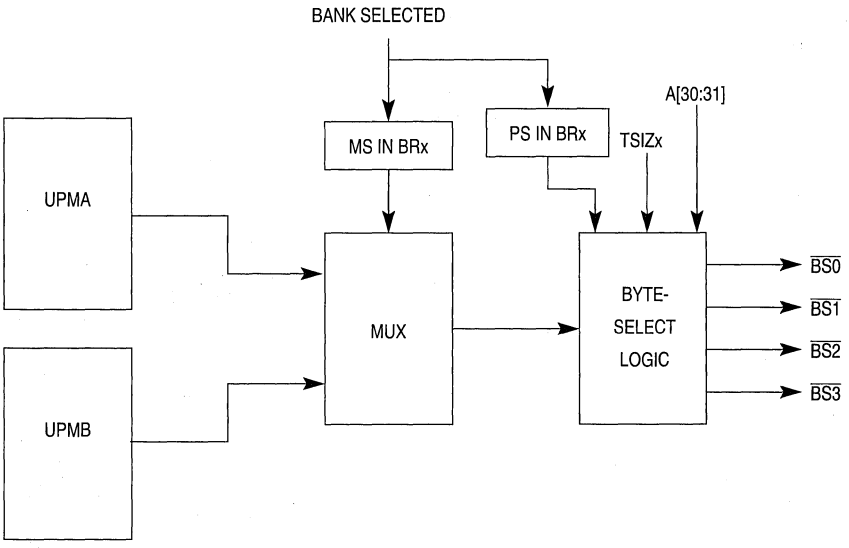


Figure 15-29. $\overline{BSx}$ Signal Selection

The uppermost byte select ($\overline{BS0}$) indicates that the most-significant eight bits of the data bus (D0-7) contain valid data during a cycle and the upper-middle byte-select ($\overline{BS1}$) indicates that the upper-middle eight bits of the data bus (D[8-15]) contain valid data during a cycle. The lower-middle byte-select ($\overline{BS2}$) indicates that the lower-middle eight bits of the data bus (D[16-23]) contain valid data during a cycle and the lowest byte-select ($\overline{BS3}$) indicates that the least-significant eight bits of the data bus (D[24-31]) contain valid data during a cycle. The manner in which the $\overline{BSx}$ signals affect 32-, 16-, and 8-bit accesses is shown in Table 15-5. It should be noted that for a periodic timer request and a memory command request, the $\overline{BSx}$ signals are only determined by the port size of the bank.

Table 15-5. Enabling Byte-Selects

TRANSFER SIZE	TSIZx		ADDRESS		32-BIT PORT SIZE				16-BIT PORT SIZE				8-BIT PORT SIZE			
			A30	A31	BS0	BS1	BS2	BS3	BS0	BS1	BS2	BS3	BS0	BS1	BS2	BS3
Byte	0	1	0	0	X				X				X			
	0	1	0	1		X				X			X			
	0	1	1	0			X		X				X			
	0	1	1	1				X		X			X			
Half-Word	1	0	0	0	X	X			X	X			X			
	1	0	1	0			X	X	X	X			X			
Word	0	0	0	0	X	X	X	X	X	X			X			

15.5.4.2.4 General-Purpose Signals. The general-purpose (GPL[1:5]) signals have two bits in the RAM word that define the logical value of the signal to be changed at the falling edge of GCLK1 or GCLK2. GPL0 has two 2-bit fields that perform the same function with additional phase control. GPL5 and GPL0 offer the following enhancements beyond the other GPLx signals:

- GPL5 can be controlled during phase 4 of the previous clock cycle according to the value of G5LS.

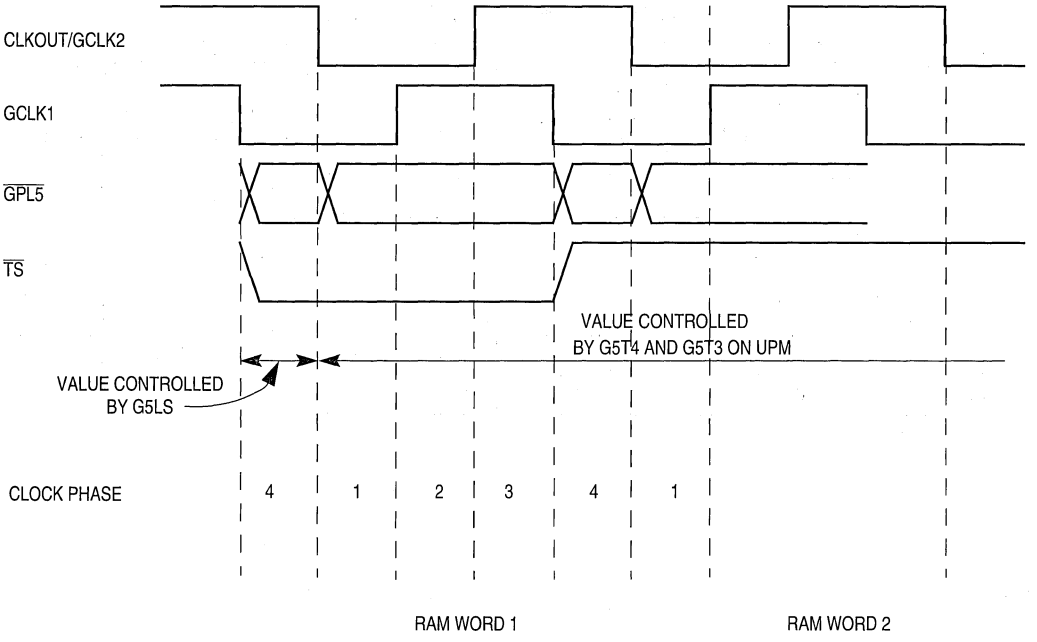


Figure 15-30. Early GPL5 Control

- GPL0 can be controlled by an address line that is specified in the G0CLx field of the MxMR. To use this feature, you must set the G0H and G0L fields in the RAM word. For example, if you have a SIMM with multiple banks, this address line can be used to switch between banks.

15.5.4.2.5 Loop Control. The LOOP bit in the RAM word allows you to run a repetitive program fragment a specific number of times. The first time the LOOP bit is asserted in a RAM word, the memory controller recognizes it as a Loop Start Word. At this time, the memory loop counter is loaded with the corresponding contents of the LOOP bit shown in Table 15-6. The next RAM word encountered with the LOOP bit set is recognized as a Loop End Word. At this time, the memory loop counter is decremented by one.

Continued loop execution depends on the memory loop counter. If the loop counter is not zero, the next RAM word executed is the Loop Start Word. Otherwise, the next RAM word executed is the word following the Loop End Word. Loops can be sequentially executed, but not nested.

Table 15-6. MxMR Loop Bit Usage

REQUEST SERVICED	LOOP BIT
Read Single Beat Cycle	RLFx
Read Burst Cycle	RLFx
Write Single Beat Cycle	WLFx
Write Burst Cycle	WLFx
Periodic Timer Expired	TLFx

15.5.4.2.6 Exception Handling. When an access to a memory device is initiated by the MPC823 under UPM control, the external device may assert the $\overline{\text{TEA}}$, $\overline{\text{SRESET}}$, or $\overline{\text{HRESET}}$ signal. An exception occurs when one of these signals is asserted by an external device and the MPC823 begins closing the memory cycle transfer. When an exception is recognized and the EXEN bit is set in the RAM word, the next RAM word will branch to the special exception start address (EXS). See Table 15-4 for more information. You should provide an exception handler to handle output signals controlled by the UPM. For DRAM control, you would provide a handler that would negate $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ to prevent data corruption. The EXEN bit is similar to an exception mask in that if it is 0, then it defers the exception and continues executing. If a RAM word is encountered with the EXEN bit set, the UPM performs a branch to the exception start address. When the branch to the exception start address is performed, the UPM continues reading until the LAST bit is set in the RAM word.

15.5.4.2.7 Address Multiplexing. You can control which address signals go to the external bus. The AMA and AMB fields of the MxMR control how the address signals are multiplexed. The SAM bit in the option register determines the address multiplexing for the first clock cycle. The AMX field in the RAM word determines the multiplexing for subsequent clock cycles. The lower address pins can be multiplexed between the internal upper or lower address signals. The SAM bit outputs the upper address signals and the AMX field outputs the lower address signals. Using the AMX field, you can output the contents of the memory address register (MAR) on the address pins. See Table 15-6 for general configuration. Table 15-7 shows how the AMA and AMB fields can be defined to interface with a wide range of DRAM modules. Figure 15-31 illustrates address multiplex timing.

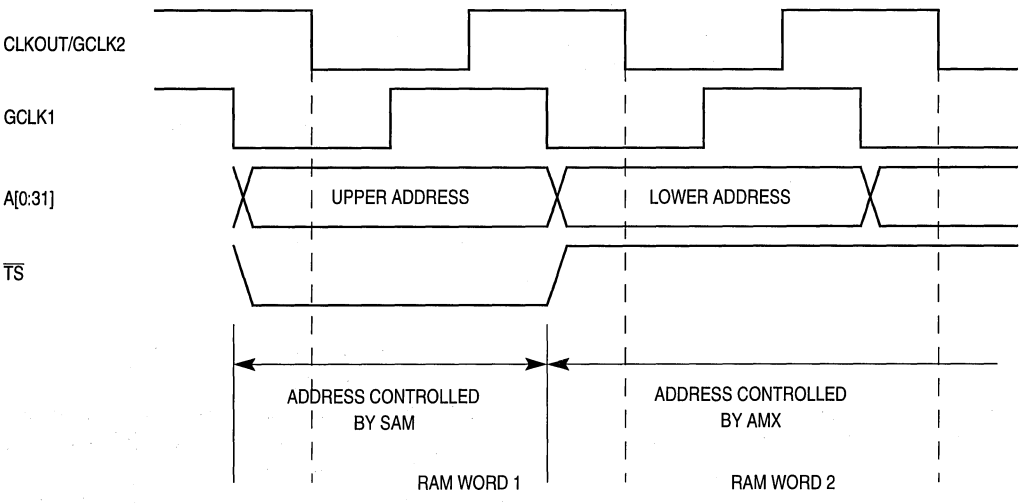


Figure 15-31. Address Multiplex Timing

Table 15-7. Address Multiplexing

		PINS															
	AMA/AMB	A16	A17	A18	A19	A20	A21	A22	A23	A24	A25	A26	A27	A28	A29	A30	A31
SIGNALS	000	RES	RES	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20	A21	A22	A23
	001	RES	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20	A21	A22
	010	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20	A21
	011	RES	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20
	100	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19
	101	RES	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18

Table 15-8. AMA/AMB Definition For DRAM Interface

DATA BUS WIDTH	MEMORY SIZE	NUMBER OF DRAM ROW ADDRESS PINS	NUMBER OF DRAM COLUMN ADDRESS PINS	MPC823 ADDRESS PIN CONNECTION	AMA/AMB IN MxMR
8 Bits	64K	8	8	A24 - A31	000
	128K	9		A23 - A31	
	256K	10		A22 - A31	
	512K	11		A21 - A31	
	1M	12		A20 - A31	
	2M	13		A19 - A31	
	4M	14		A18 - A31	
	256K	9	9	A23 - A31	001
	512K	10		A22 - A31	
	1M	11		A21 - A31	
	2M	12		A20 - A31	
	4M	13		A19 - A31	
	8M	14		A18 - A31	
	16M	15		A17 - A31	
	1M	10	10	A22 - A31	010
	2M	11		A21 - A31	
	4M	12		A20 - A31	
	8M	13		A19 - A31	
	16M	14		A18 - A31	
	32M	15		A17 - A31	
	64M	16		A16 - A31	
	4M	11	11	A21 - A31	011
	8M	12		A20 - A31	
	16M	13		A19 - A31	
	32M	14		A18 - A31	
	64M	15		A17 - A31	
	16M	12	12	A20 - A31	100
	32M	13		A19 - A31	
	64M	14		A18 - A31	
	128M	15		A17 - A31	
	256M	16		A16 - A31	

Table 15-8. AMA/AMB Definition For DRAM Interface (Continued)

DATA BUS WIDTH	MEMORY SIZE	NUMBER OF DRAM ROW ADDRESS PINS	NUMBER OF DRAM COLUMN ADDRESS PINS	MPC823 ADDRESS PIN CONNECTION	AMA/AMB IN MxMR
8 Bits	64M	13	13	A19 - A31	101
	128M	14		A18 - A31	
	256M	15		A17 - A31	
16 Bits	128K	8	8	A23 - A30	000
	256K	9		A22 - A30	
	512K	10		A21 - A30	
	1M	11		A20 - A30	
	2M	12		A19 - A30	
	4M	13		A18 - A30	
	512K	9	9	A22 - A30	001
	1M	10		A21 - A30	
	2M	11		A20 - A30	
	4M	12		A19 - A30	
	8M	13		A18 - A30	
	16M	14		A17 - A30	
	2M	10	10	A21 - A30	010
	4M	11		A20 - A30	
	8M	12		A19 - A30	
	16M	13		A18 - A30	
	32M	14		A17 - A30	
	64M	15		A16 - A30	
	8M	11	11	A20 - A30	011
	16M	12		A19 - A30	
	32M	13		A18 - A30	
	64M	14		A17 - A30	
	32M	12	12	A19 - A30	100
	64M	13		A18 - A30	
	128M	14		A17 - A30	
	256M	15		A16 - A30	
	128M	13	13	A18 - A30	101
	256M	13		A17 - A30	

Table 15-8. AMA/AMB Definition For DRAM Interface (Continued)

DATA BUS WIDTH	MEMORY SIZE	NUMBER OF DRAM ROW ADDRESS PINS	NUMBER OF DRAM COLUMN ADDRESS PINS	MPC823 ADDRESS PIN CONNECTION	AMA/AMB IN MxMR
32 Bits	256K	8	8	A22 - A29	000
	512K	9		A21 - A29	
	1M	10		A20 - A29	
	2M	11		A19 - A29	
	4M	12		A18 - A29	
	1M	9	9	A21 - A29	001
	2M	10		A20 - A29	
	4M	11		A19 - A29	
	8M	12		A18 - A29	
	16M	13		A17 - A29	
	4M	10	10	A20 - A29	010
	8M	11		A19 - A29	
	16M	12		A18 - A29	
	32M	13		A17 - A29	
	64M	14		A16 - A29	
	16M	11	11	A19 - A29	011
	32M	12		A18 - A29	
	64M	13		A17 - A29	
	64M	12	12	A18 - A29	100
	128M	13		A17 - A29	
	256M	14		A16 - A29	
	256M	13	13	A17 - A29	101

15.5.4.2.8 Transfer Acknowledge and Data Sample Control. During a memory access, the UTA bit of the RAM word controls the state of the $\overline{TA}$ signal sampled by the bus master. The $\overline{TA}$ signal is driven on the rising edge of GCLK2. When a read access is handled by the UPM and the UTA bit is 0, the value of the DLT3 bit in the same RAM word indicates when the data input is sampled by the bus master, assuming that the GPLx4DIS bit is set in the MxMR. Figure 15-32 illustrates the data sampling that is controlled by the UPM.

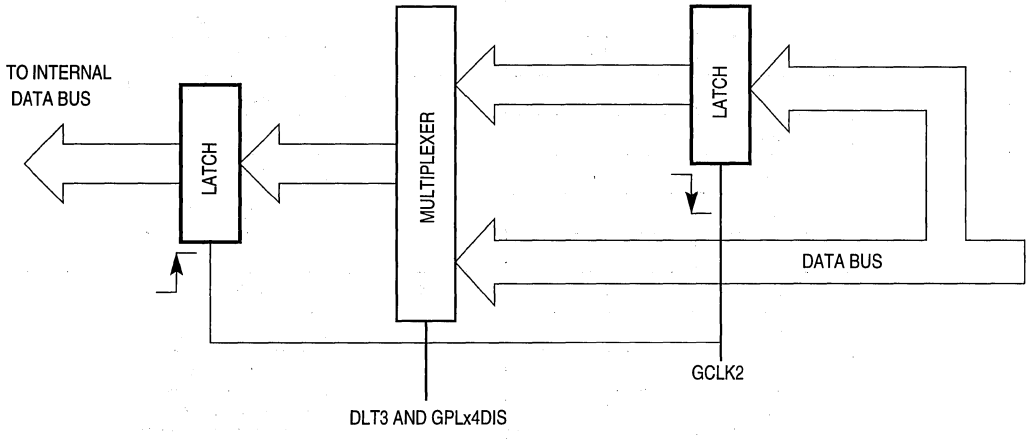


Figure 15-32. UPM Read Access Data Sampling

15.5.4.2.9 Disable Timer Mechanism. The disable timer associated with each UPM allows you to guarantee a minimum time between two successive accesses to the same memory bank. This feature is critical when DRAM requires a $\overline{RAS}$ precharge time. The TODT bit in the RAM word turns the timer on to prevent another UPM access to the same bank until the timer expires.

The disable timer does not affect memory accesses to different banks. If the timing specified by the UPM RAM word is less than the disable timer period, the access to the next bank will conflict with the current bank access. To avoid conflicts between different banks using the same UPM, the number of words in the RAM array must be equal to or greater than the period defined in the DSx field of the MxMR.

15.5.4.2.10 Last Word. When the LAST bit is read in a RAM word, the highest priority pending request (if any) is serviced immediately in the external memory transactions. If the disable timer is activated, the bus will be idle for a number of clock cycles as specified in the DSx field of the MxMR.

15.5.5 The Wait Mechanism

If the UPM reads a RAM word with the WAEN bit set, the external UPWAITx signal is sampled and synchronized by the memory controller and the current request is frozen. If the WAEN bit is asserted, the logical value of the external signals are frozen to the value defined in the last RAM word accessed and the RAM address increment is disabled until the UPWAITx signal is negated. This allows wait states to be inserted as required by an external device through an external signal. A memory disable timer is associated with each UPM. This timer counts down to zero starting at the value programmed in the DSx field in the MxMR.

15.5.5.1 INTERNAL AND EXTERNAL SYNCHRONOUS MASTER. Figure 15-33 illustrates how the WAEN bit in the word read by the UPM and the UPWAITx signal is used to hold the UPM in a particular state until the UPWAITx signal is negated. As illustrated in Figure 15-33, the $\overline{CSx}$ and $\overline{GPL1}$ states (C12 and F) and the WAEN value (CC) are frozen until the UPWAITx signal is recognized as deasserted.

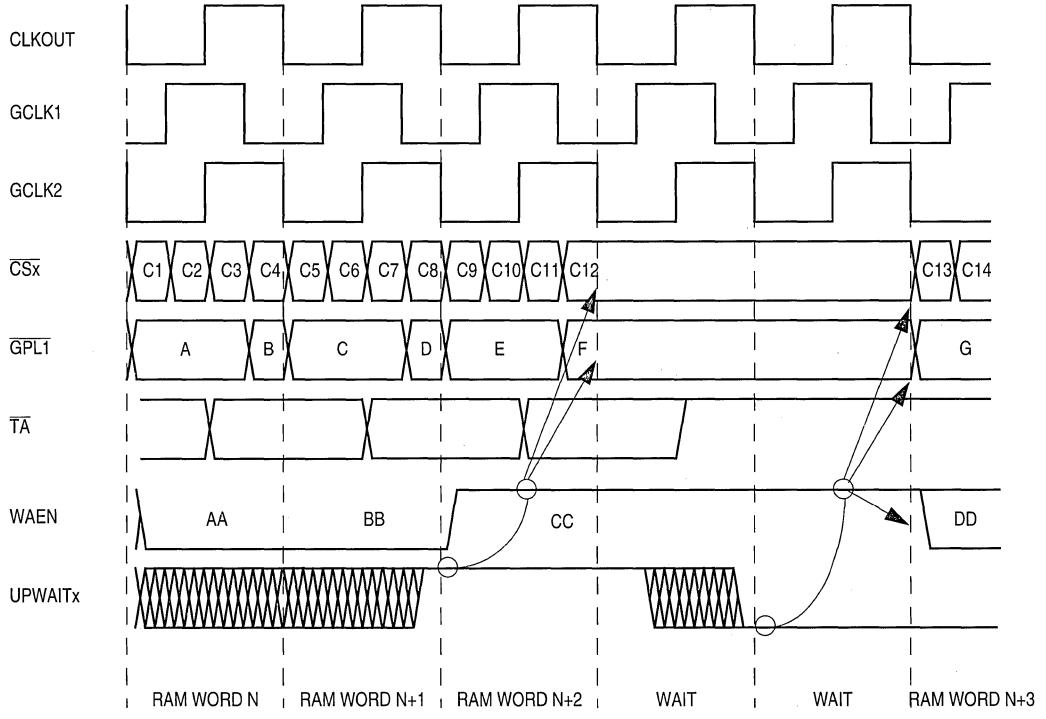


Figure 15-33. Wait Mechanism Timing For Internal and External Synchronous Masters

The UPWAITx signal is sampled at the falling edge of the CLKOUT. If the signal is asserted and the WAEN bit is set in the current RAM word, the UPM is frozen until the UPWAITx signal is recognized as negated. The value of the external pins driven by the UPM remains as indicated in the word previously read by the UPM. When the UPWAITx signal is negated, the UPM continues with its normal functions. Notice that during the wait cycles, the $\overline{TA}$ signal is negated by the UPM.

15.5.5.2 EXTERNAL ASYNCHRONOUS MASTER. The UPM supports asynchronous external masters using the $\overline{AS}$ signal. When you use an external asynchronous master, the $\overline{AS}$ signal behaves like the UPWAITx signal. The UPM enters a wait state if the $\overline{AS}$ signal is asserted and the WAEN bit is set in the current RAM word. As illustrated in Figure 15-34, the $\overline{CSx}$ and $\overline{GPL1}$ states (C12 and F) and the WAEN value (CC) are frozen until the $\overline{AS}$ signal is recognized as deasserted. The $\overline{TA}$ signal that is driven by the UPM remains in its previous value until the $\overline{AS}$ signal is negated. The state of the external pins driven by the UPM remains as indicated in the word previously read by the UPM.

To exit a wait state, the $\overline{AS}$ signal should be negated. When $\overline{AS}$ is negated, all external signals controlled by the UPM are driven high. The external signals are driven in this state until the LAST bit is set in a RAM word. The TODT bit is only relevant in the words read by the UPM after the $\overline{AS}$ signal is negated. Refer to **Section 15.6 External Master Support** for more information.

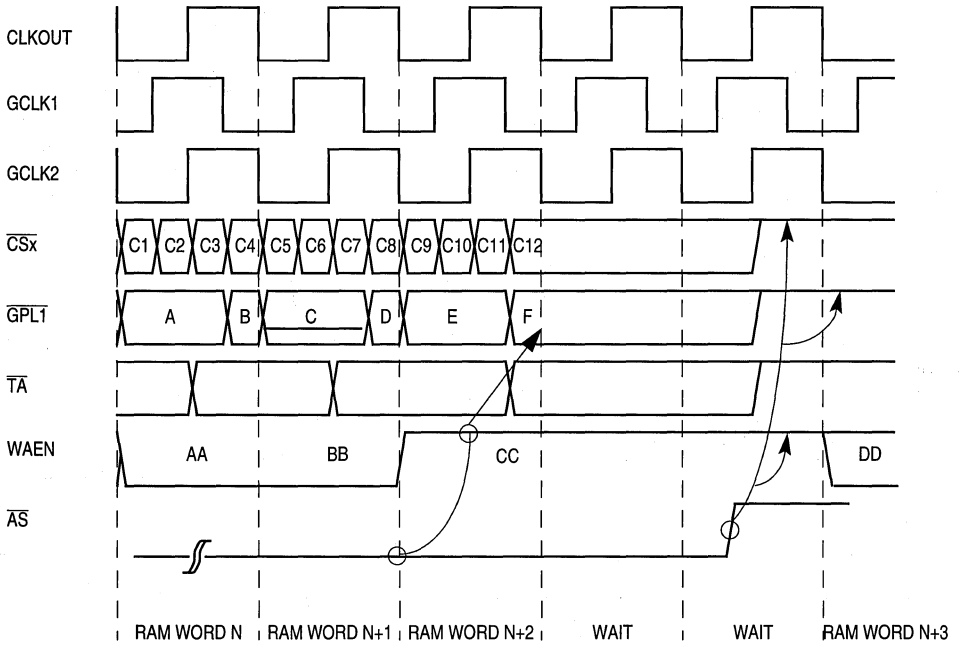


Figure 15-34. Wait Mechanism Timing For An External Asynchronous Master

15.5.5.3 HANDLING VARIABLE ACCESS TIME AND SLOW DEVICES. The memory controller provides two different mechanisms to interface with slave devices that either are very slow or cannot guarantee a predefined access time—the wait state and the external TA signal. These devices can be divided into two main types:

- Variable access time devices (FIFO, hierarchical bus interface, dual-port memory devices)
- Slow devices (access time is greater than the maximum allowed by the UPM)

The wait mechanism is used only in accesses that are controlled by the UPM. The GPLx4DIS bit of the MxMR enables this mechanism. The external $\overline{TA}$ mechanism is used only in accesses that are controlled by the GPCM. The SETA bit in the option register specifies whether the $\overline{TA}$ is generated internally or externally.

15.5.5.3.1 Hierarchical Bus Interface Example. On the local bus, the CPU initiates a read cycle that addresses the main storage connected to the system bus. The hierarchical bus interface accepts the local bus request and generates a read cycle on the system bus. You cannot foresee when the data will be valid to be latched by the CPU since the system bus may be occupied by the DMA.

- The Wait Solution—The external module signals to the memory controller that the data is not ready yet by asserting the UPWAITx signal. The memory controller synchronized this signal since it is an asynchronous signal. As a result of the UPWAITx signal being asserted, the UPM will enter a freeze mode at the falling edge of the CLKOUT upon encountering the WAEN bit being set in the UPM word. The UPM will remain in that state until the UPWAITx signal is asserted. After the negation of UPWAITx, the UPM will continue executing from the next entry to the end of the pattern (LAST bit is set).
- The External $\overline{TA}$ Solution—The bus interface module signals to the memory controller when it can sample the data by asserting the synchronous $\overline{TA}$ signal.

15.5.5.3.2 Slow Device Interface Example. The CPU initiates a read cycle from slow devices whose access time is greater than the maximum allowed by your programming model.

- The Wait Solution—The CPU generates a read access from the slow device. The device will react by asserting the UPWAITx signal as long as the data is not ready. The CPU will sample the data only after the negation of the UPWAITx signal.
- The External $\overline{TA}$ Solution—The CPU generates a read access from the slow device. When it is ready, the device is responsible for generating the synchronous $\overline{TA}$ signal.

15.6 EXTERNAL MASTER SUPPORT

The memory controller supports internal and external bus masters. Accesses that originate from the core, CPM, and LCD controller are considered internal and those initiated by an external bus master are external. External bus master support can be enabled in the SIU module configuration register (SIUMCR), as indicated in **Section 12.12.1.1 SIU Module Configuration Register**. There are two types of external bus masters:

- Synchronous bus masters synchronize with CLKOUT and may use the MPC823 memory controller to access a slave device or bypass the memory controller to perform the slave access.
- Asynchronous bus masters use an address strobe ($\overline{AS}$) signal that handshakes with the MPC823 memory controller to access a slave device or bypass the memory controller to perform the slave access.

Synchronous masters initiate a transfer by asserting the $\overline{TS}$ signal. The A[6:31], RD/$\overline{WR}$, and TSIZx signals must be stable prior to the rising edge of CLKOUT after $\overline{TS}$ assertion and until the last $\overline{TA}$ signal is negated. Since the external master operates synchronously with the MPC823, proper setup and hold times for all inputs associated with the rising edge of CLKOUT are significant. To support synchronous mode using the memory controller, the SEME bit in the SIUMCR must be set. When the $\overline{TS}$ signal is asserted, the memory controller compares the address with each one of its defined valid banks and if a match is found, control signals to the slave device are generated and the $\overline{TA}$ signal is supplied to the external master. If the SEME bit is cleared, the memory controller is bypassed and the external synchronous master must provide control signals to the slave device. See Figure 15-35 for details.

Asynchronous masters initiate a transfer by driving the address bus and asserting the $\overline{AS}$ pin. The A[6:31] signals, together with RD/$\overline{WR}$ and TSIZx, must have a proper setup time prior to $\overline{AS}$ pin assertion. To support asynchronous mode using the memory controller, the AEME bit in the SIUMCR must be set. The memory controller synchronizes $\overline{AS}$ assertion to its internal clock and generates the control signals to the slave device. When the $\overline{AS}$ pin is synchronized, the memory controller compares the address with each one of its defined valid banks and if a match is found, control signals to the slave device are generated and the $\overline{TA}$ signal is supplied to the external master. All the control signals to the memory device and the $\overline{TA}$ signal are negated with $\overline{AS}$ pin negation. If the AEME bit is cleared, the memory controller is bypassed and the external asynchronous master must provide control signals to the slave device. In this mode, the $\overline{AS}$ pin of the MPC823 is not available as an input. See Figure 15-36 for details.

Note: When external masters access slaves on the bus, the internal AT[0:2] signals reaching the memory controller will be forced to '100'. You should make sure this access matches the AT field in the base register after it is masked by the ATM field in the option register.

The A[28:20] pins should be used to generate addresses to memory devices during burst accesses. They duplicate the value of the A[28:20] signals when an internal master initiates a transaction on the external bus. When an external master initiates a transaction on the external bus, the A[28:20] pins reflect the value of the A[28:20] pins on the first clock cycle of the memory access. On subsequent clock cycles, the behavior of the A[28:20] pins depends on the configuration of the UPM.

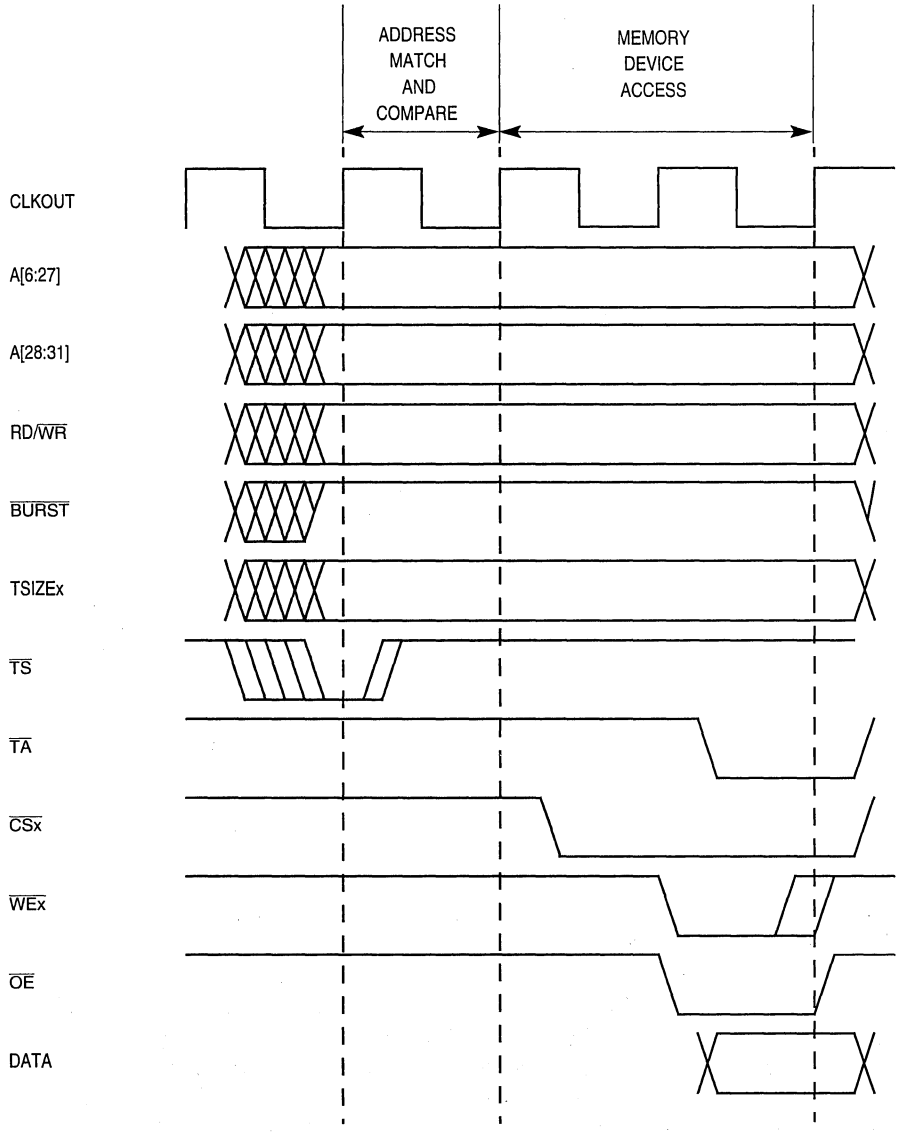


Figure 15-35. Synchronous External Master Access

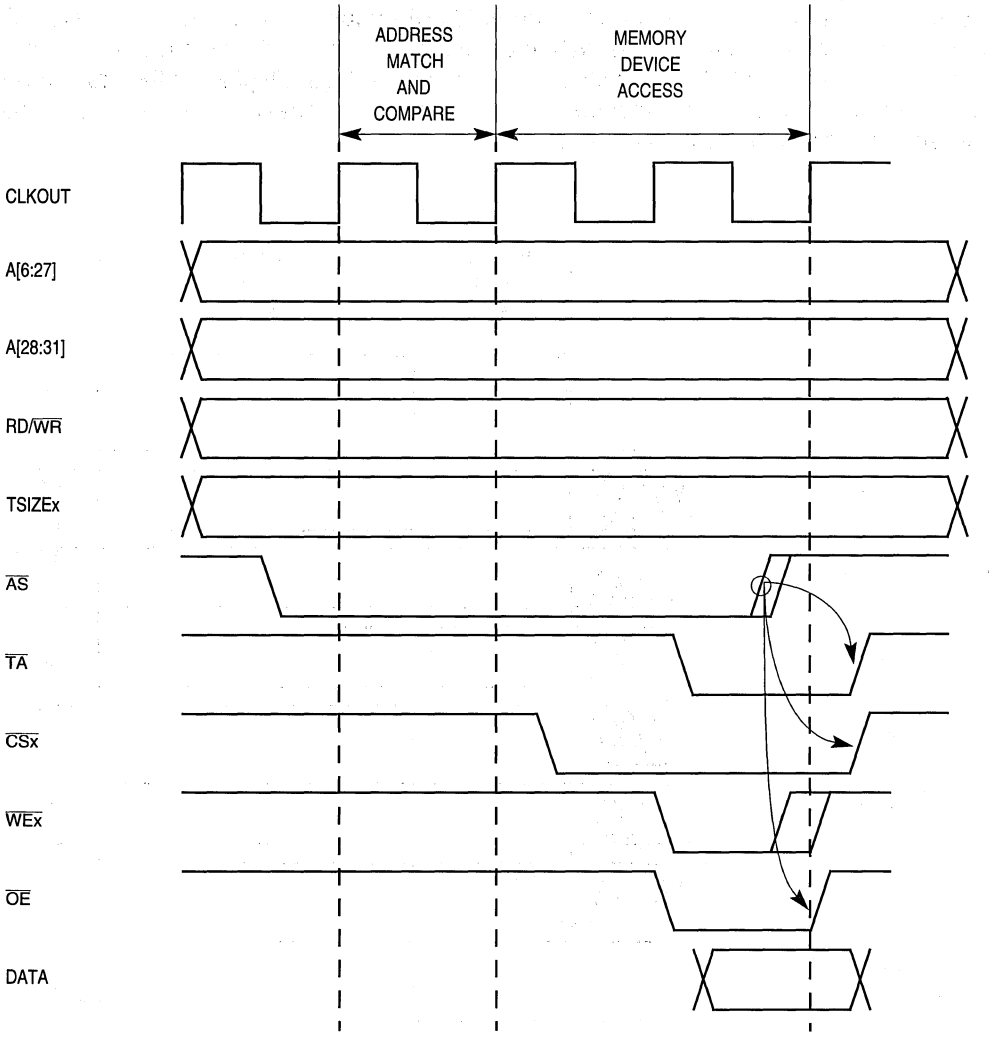


Figure 15-36. Asynchronous External Master Access

To connect to external memory devices that require address multiplexing, use the GPL_x5 pin. The state of the GPL_x5 signal logic value depends on the configuration defined in Table 15-9. The GPL_x5 pin reflects the value of the G5LS bit of the corresponding option register in the first clock cycle of the slave device access. In subsequent clock cycles, the state of GPL_x5 is determined by the G5T4 and G5T3 bits in the RAM word. If the UPMB controls the slave access, you can use the G5LA bit of the option register to select the active GPL_x5 signal. G5LS only applies to memory requests and not RAM words executed by internal/external software, exception, or memory periodic timer requests.

Table 15-9. GPL_x5 Signal (Pin) Behavior

MACHINE CONTROLLING THE MEMORY ACCESS	MACHINE CONTROLLING THE SLAVE ACCESS CLOCK CYCLE	G5LA (ORx)	G5LS (ORx)	G5T4 (RAM WORD)	G5T3 (RAM WORD)	GPL_x5 BEHAVIOR AT THE CONTROLLING CLOCK EDGE
GPCM	x	N/A	N/A	x	x	GPL_A5 and GPL_B5 do not change their value.
UPMA	First	x	0	x	x	GPL_A5 is driven low at the falling edge of GCLK1.
			1			GPL_A5 is driven high at the falling edge of GCLK1.
	Second, Third...	x	x	0	x	GPL_A5 is driven low at the falling edge of GCLK2 in the current UPM cycle.
				1	x	GPL_A5 is driven high at the falling edge of GCLK2 in the current UPM cycle.
				x	0	GPL_A5 is driven low at the falling edge of GCLK1 in the current UPM cycle.
				x	1	GPL_A5 is driven high at the falling edge of GCLK1 in the current UPM cycle.
UPMB	First	0	0	x	x	GPL_B5 is driven low at the falling edge of GCLK1.
			1			GPL_B5 is driven high at the falling edge of GCLK1.
		1	0	x	x	GPL_A5 is driven low at the falling edge of GCLK1.
			1			GPL_A5 is driven high at the falling edge of GCLK1.
	Second, Third...	0	x	0	x	GPL_B5 is driven low at the falling edge of GCLK2 in the current UPM cycle.
				1	x	GPL_B5 is driven high at the falling edge of GCLK2 in the current UPM cycle.
				x	0	GPL_B5 is driven low at the falling edge of GCLK1 in the current UPM cycle.
				x	1	GPL_B5 is driven high at the falling edge of GCLK1 in the current UPM cycle.
UPMB	Second, Third...	1	x	0	x	GPL_A5 is driven low at the falling edge of GCLK2 in the current UPM cycle.
				1	x	GPL_A5 is driven high at the falling edge of GCLK2 in the current UPM cycle.
				x	0	GPL_A5 is driven low at the falling edge of GCLK1 in the current UPM cycle.
				x	1	GPL_A5 is driven high at the falling edge of GCLK1 in the current UPM cycle.

15.6.1 External Master Examples

A synchronous example interconnection in which an external master and the MPC823 can both share access to a DRAM bank is illustrated in Figure 15-37. Notice that $\overline{CS1}$, UPMA, and $\overline{GPL\_A5}$ were chosen to assist in the control of DRAM bank accesses. To perform burst accesses initiated by the external master or MPC823 using this configuration, the A[28:30] signals are connected to the multiplexer controlled by $\overline{GPL\_A5}$. Figure 15-38 illustrates the timing behavior of control signals when an external master to a DRAM bank initiates a burst read access. The state of the $\overline{GPL\_A5}$ pin in the first clock cycle of the memory device access is determined by the value of the G5LS bit in the corresponding option register. In this example, the accessed critical word is addressed at A[28:29] = 10, which then increments and wraps around to the word before the critical word (01) for subsequent beats of this burst access.

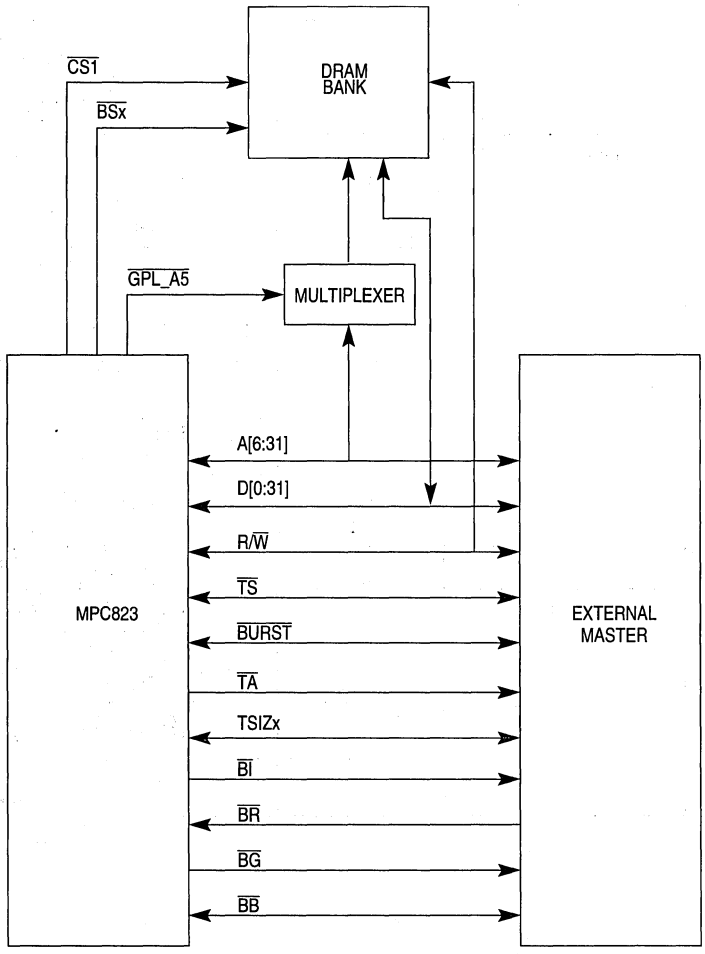
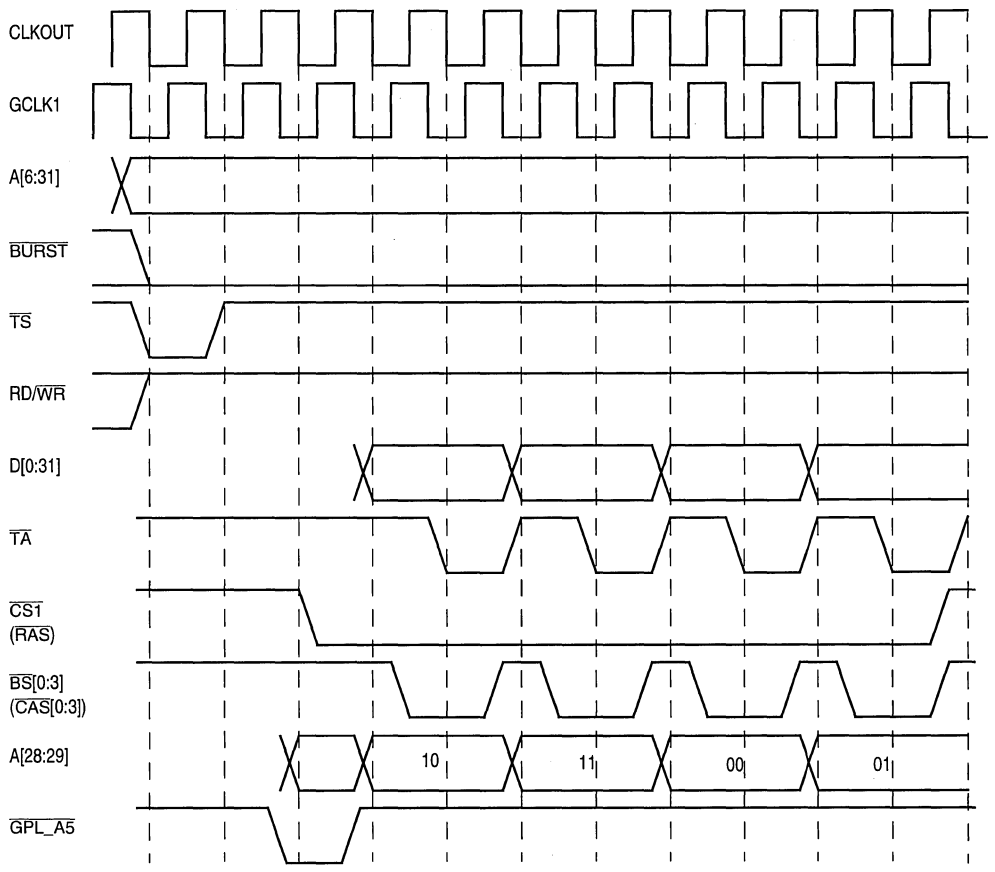


Figure 15-37. Synchronous External Master Interconnect Example



cst4	(Bit 0)	0	0	0	0	0	0	0	0	0
cst1	(Bit 1)	0	0	0	0	0	0	0	0	0
cst2	(Bit 2)	0	0	0	0	0	0	0	0	1
cst3	(Bit 3)	0	0	0	0	0	0	0	0	1
bst4	(Bit 4)	1	1	0	1	0	1	0	1	0
bst1	(Bit 5)	1	0	0	0	0	0	0	0	0
bst2	(Bit 6)	1	0	1	0	1	0	1	0	1
bst3	(Bit 7)	1	0	1	0	1	0	1	0	1
g0l0	(Bit 8)									
≈										
g5t4	(Bit 20)	0	1	1	1	1	1	1	1	1
g5t3	(Bit 21)	0	0	0	0	0	0	0	0	0
-	(Bit 22)									
-	(Bit 23)									
loop	(Bit 24)	0	0	0	0	0	0	0	0	0
exen	(Bit 25)	0	0	1	0	1	0	1	0	0
amx0	(Bit 26)	0	0	0	0	0	0	0	0	0
amx1	(Bit 27)	0	0	0	0	0	0	0	0	0
na	(Bit 28)	0	0	1	0	1	0	1	0	0
uta	(Bit 29)	1	0	1	0	1	0	1	0	1
todt	(Bit 30)	0	0	0	0	0	0	0	0	1
last	(Bit 31)	0	0	0	0	0	0	0	0	1
		RBS	**RBS+1**	**RBS+2**	**RBS+3**	**RBS+4**	**RBS+5**	**RBS+6**	**RBS+7**	**RBS+8**

Figure 15-38. Synchronous External Master–Burst Read Access To Page Mode DRAM

An asynchronous example interconnection in which an external master and the MPC823 can both share access to a DRAM bank is illustrated in Figure 15-39. Notice that CS1, UPMA, and GPL_A5 were chosen to assist in the control of DRAM bank accesses. Figure 15-40 illustrates the timing behavior of the GPL_A5 and other control signals when an external master to a DRAM bank initiates a single beat read access. The state of the GPL_A5 pin in the first clock cycle of the memory device access is determined by the value of the G5LS bit in the corresponding option register.

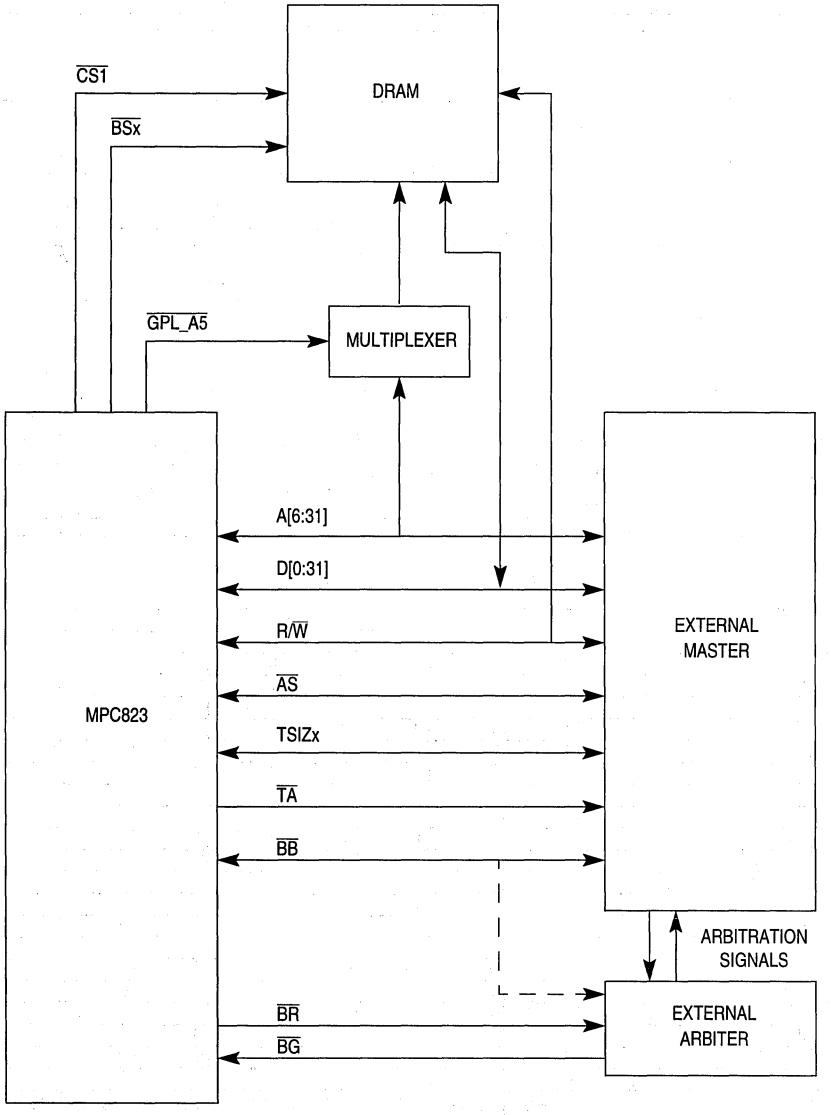
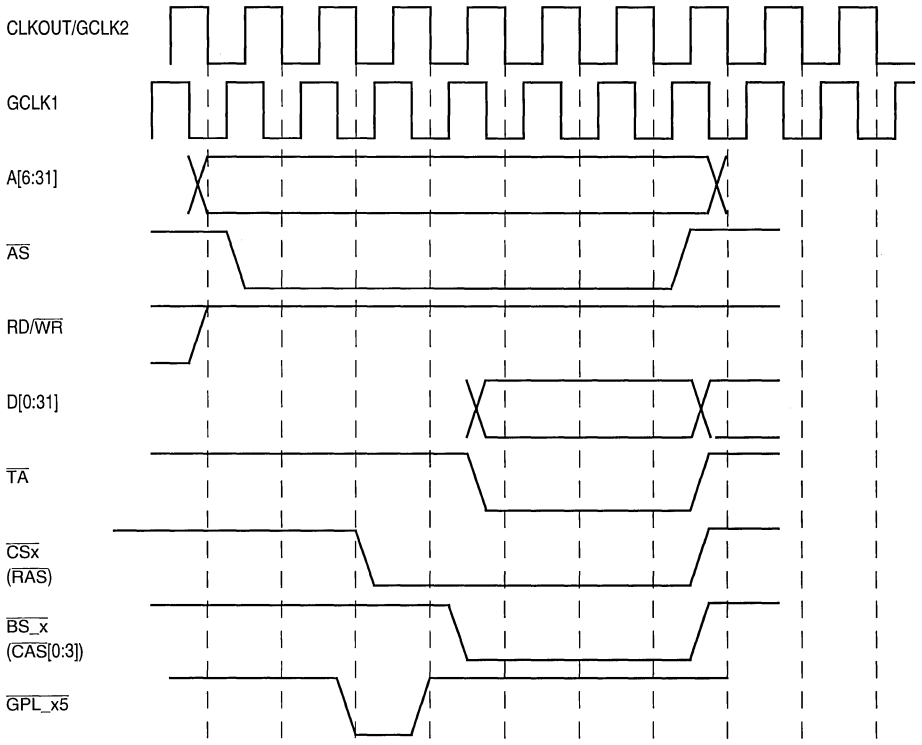


Figure 15-39. Asynchronous External Master Interconnect Example



cst4	Bit 0	0	0	0	0	0
cst1	Bit 1	0	0			0
cst2	Bit 2	0	0			1
cst3	Bit 3	0	0			1
bst4	Bit 4	1	1	0	0	0
bst1	Bit 5	1	0			0
bst2	Bit 6	1	0			1
bst3	Bit 7	1	0			1
≈						
g4t4	Bit 18					
g4t3	Bit 19					
g5t4	Bit 20			1	1	
g5t3	Bit 21	0	1			1
-	Bit 22					
-	Bit 23					
loop	Bit 24	0	0	0	0	0
exen	Bit 25	0	0	0	0	0
amx0	Bit 26	0	0	0	0	1
amx1	Bit 27	0	0	0	0	0
na	Bit 28	0	0	0	0	0
uta	Bit 29	1	0	0	0	1
todt	Bit 30	0	0	0	0	1
last	Bit 31	0	0	0	0	1
		RSS	RSS+1	WAIT	WAIT	RSS+2

Figure 15-40. Asynchronous External Master Timing Example

15.7 MEMORY SYSTEM INTERFACE EXAMPLES

The following examples illustrate how to connect and set up the UPM RAM array for two types of DRAM—page mode DRAM and page mode extended data out DRAM. The values used in these examples apply to either UPMA or UPMB. UPMA is used in the page mode example and UPMB is used in the extended data out example.

15.7.1 Page Mode DRAM Interface Example

The configuration for a 1M, 32-bit wide memory system using four 256K x 8-bit DRAMs is illustrated in Figure 15-41. Also shown is the physical connection between UPMA and the page mode DRAM. The $\overline{CS1}$ signal is connected to all the $\overline{RAS}$ signals and controlled by the base register. The $\overline{BS\_A}$[0:3] signals are mapped one-to-one to each of the four DRAMs and are controlled by the UPM RAM word. The refresh rate is calculated based on a 25MHz baud rate generator clock and the DRAM that requires a 512-cycle refresh every 8ms.

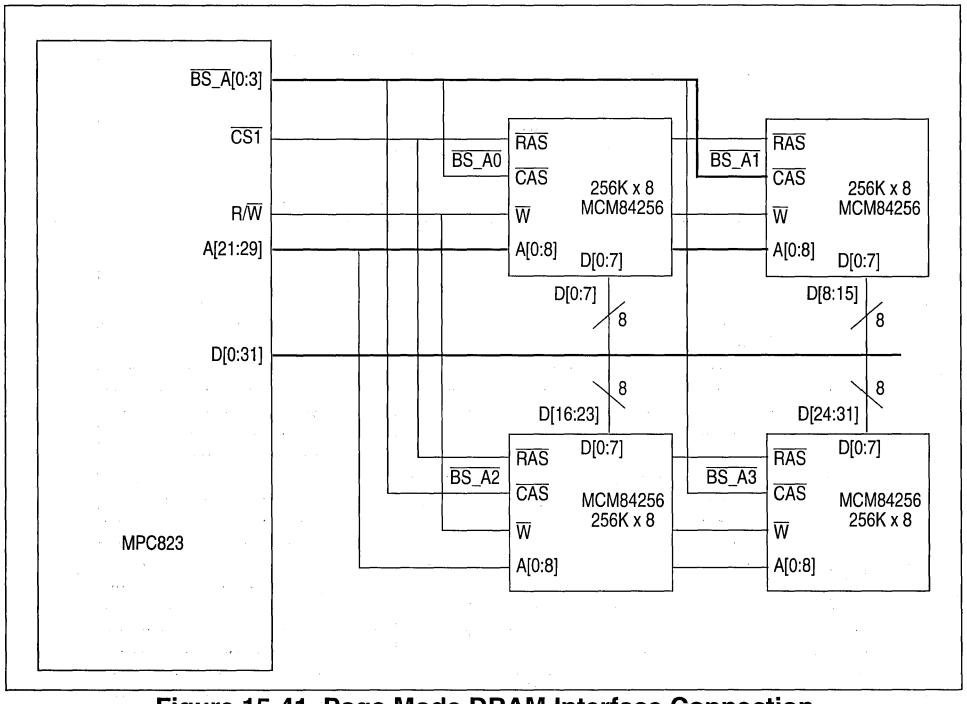


Figure 15-41. Page Mode DRAM Interface Connection

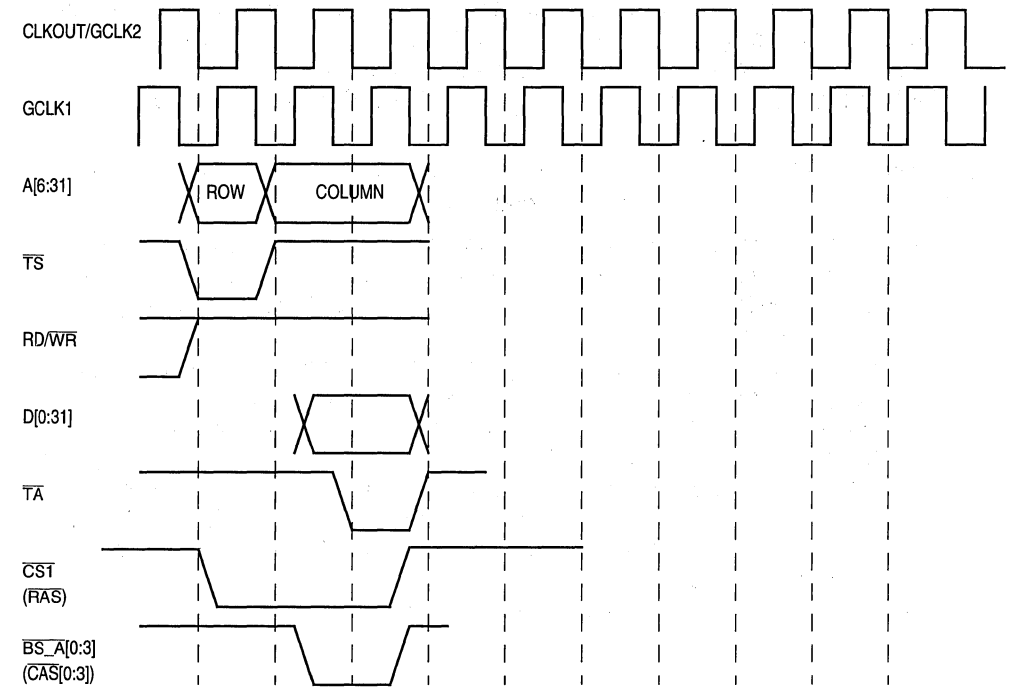
Follow these steps to configure a system for page mode DRAM:

1. Determine the system architecture, which includes the MPC823 and the memory system as shown in the example in Figure 15-41.
2. Use the blank worksheet in Figure 15-58 to draw the timing diagrams for all the memory cycles associated with your architecture. You can also use as a reference the various other timing diagrams in Figure 15-42 through Figure 15-50.

3. Translate the timing diagrams into RAM words for each type of memory access. The bottom half of the figures represent the RAM array contents that handle each of the possible cycles and each column represents a different word in the RAM array. A blank cell in the figures indicates a "don't care" bit, which is typically programmed to logic 1 to conserve power.
4. Define the UPMA (or UPMB) parameters that control the memory system in the following sequence. For additional details, see Table 15-10.
   - — Program the RAM array using the memory command register (MCR) and memory data register (MDR). The RAM word must be written into the MDR before you issue the **WRITE** command to the MCR. Repeat this step for all RAM word entries.
   - — Initialize the option and base registers of the specific bank according to the address mapping of the DRAM device you have chosen.
   - — Use the MS field of the option register to select the machine you have chosen to control the cycles. Notice that the SAM bit in the option register determines address multiplexing for the first clock cycle and subsequent cycles are controlled by the UPM RAM words. Also notice that the AMX field in the UPM RAM word controls the address multiplexing for the next clock cycle rather than the current cycle.
   - — Program the MAMR to select the number of columns and refresh timer parameters.

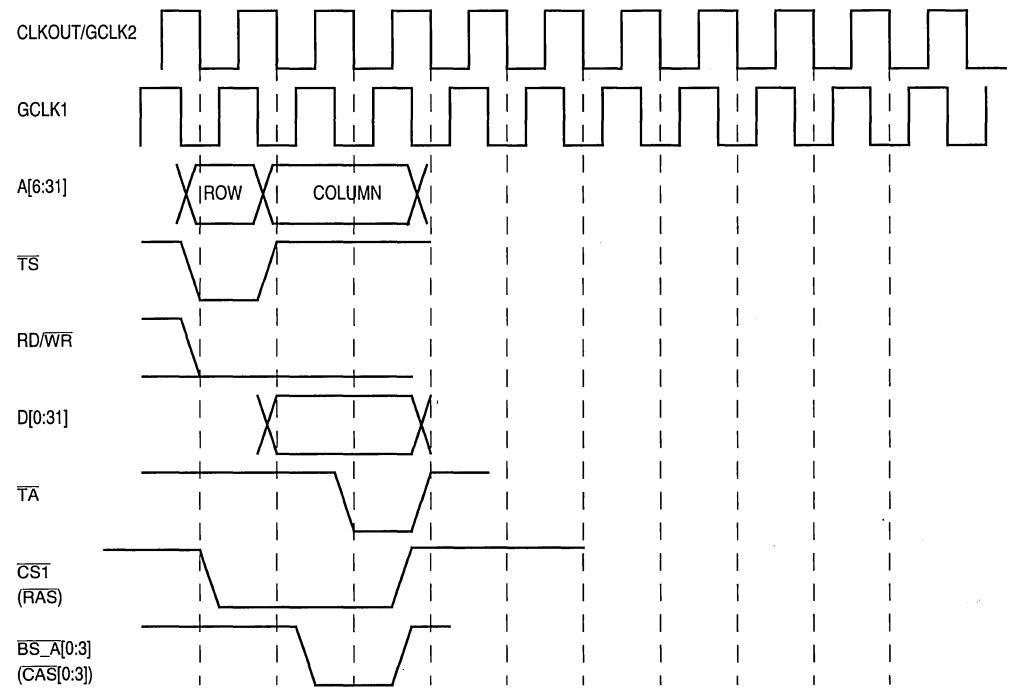
Table 15-10. UPMA Register Settings

FIELD	REGISTER	VALUE	COMMENTS
MS	BR1	10	Selects UPMA
PS	BR1	00	Selects 32-Bit Bus Width
WP	BR1	0	Allows Read and Write Accesses
PTP	MPTPR	00000010	Prescaler Divided by Two
PTA	MAMR	00001100	15.6µs at a 25MHz Clock
PTAE	MAMR	1	Enables Periodic Timer A
AMA	MAMR	001	Selects Nine Column Address Pins
DSA	MAMR	01	Selects Two Disable Timer Clock Cycles
GPLA4DIS	MAMR	0	Disables the UPWAITA Signal
RLFA	MAMR	0011	Selects Three Loop Iterations for Read
WLFA	MAMR	0011	Selects Three Loop Iterations for Write
SAM	OR1	1	Selects Column Address on First Cycle
BI	OR1	0	Supports Burst Accesses



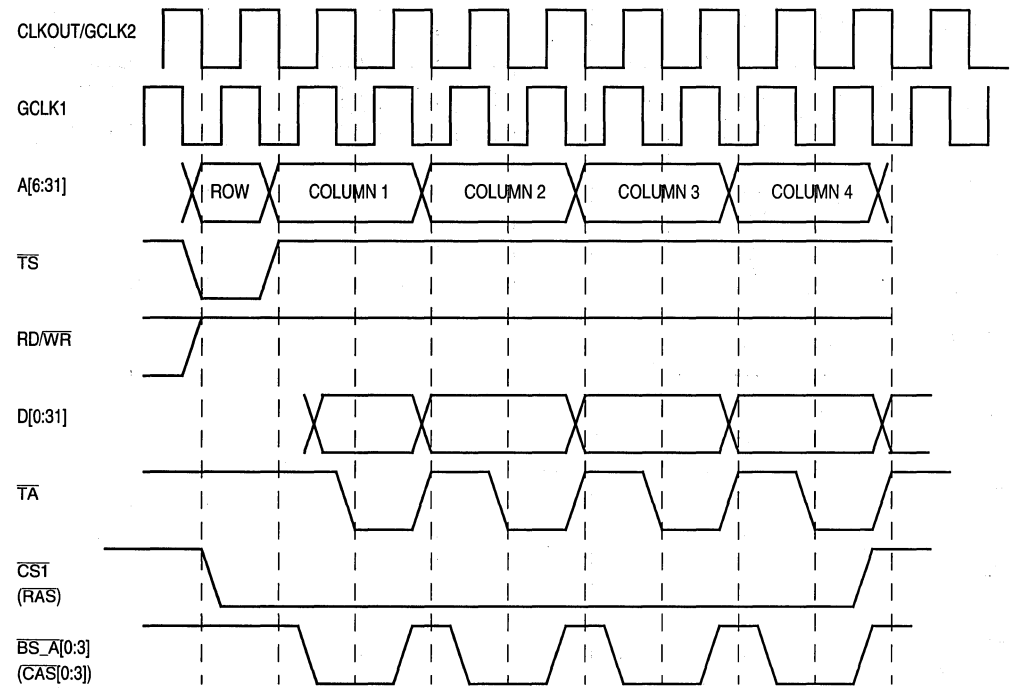
cst4	Bit 0	0	0	0
cst1	Bit 1	0	0	0
cst2	Bit 2	0	0	1
cst3	Bit 3	0	0	1
bst4	Bit 4	1	1	0
bst1	Bit 5	1	0	0
bst2	Bit 6	1	0	1
bst3	Bit 7	1	0	1
g0l0	Bit 8			
g0l1	Bit 9			
g0h0	Bit 10			
g0h1	Bit 11			
g1t4	Bit 12			
g1t3	Bit 13			
g2t4	Bit 14			
g2t3	Bit 15			
g3t4	Bit 16			
g3t3	Bit 17			
g4t4	Bit 18			
g4t3	Bit 19			
g5t4	Bit 20			
g5t3	Bit 21			
-	Bit 22			
-	Bit 23			
loop	Bit 24	0	0	0
exen	Bit 25	0	0	0
amx0	Bit 26	0	0	1
amx1	Bit 27	0	0	0
na	Bit 28	0	0	0
uta	Bit 29	1	0	1
todt	Bit 30	0	0	1
last	Bit 31	0	0	1
		RSS	**RSS+1**	**RSS+2**

Figure 15-42. Single Beat Read Access To Page Mode DRAM



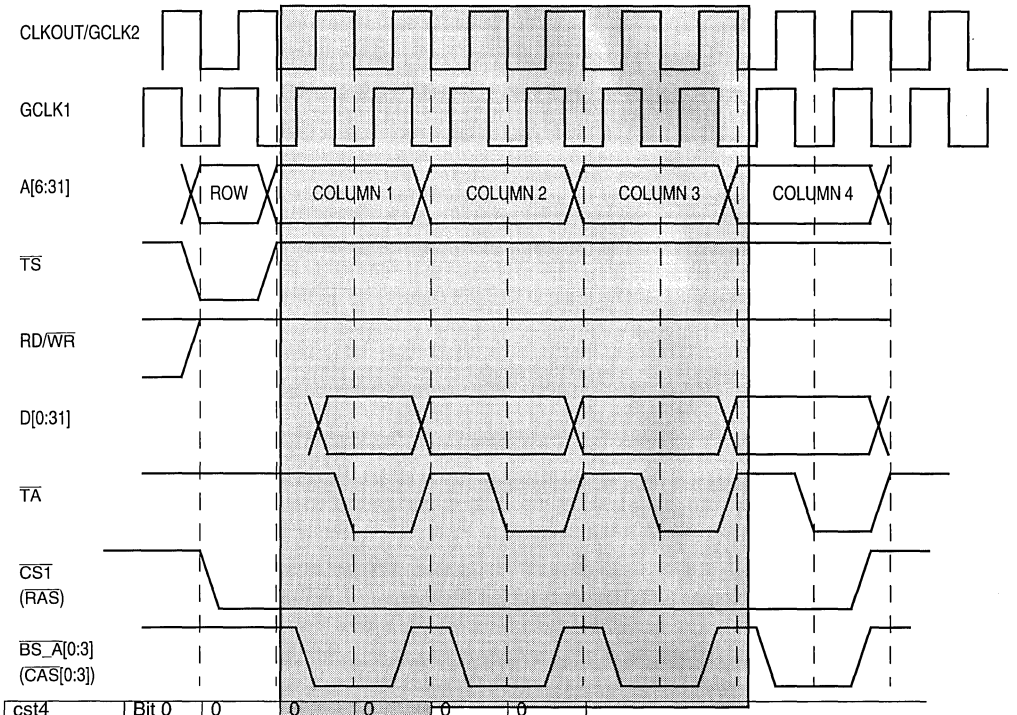
cst4	Bit 0	0	0	0
cst1	Bit 1	0	0	0
cst2	Bit 2	0	0	1
cst3	Bit 3	0	0	1
bst4	Bit 4	1	1	0
bst1	Bit 5	1	0	0
bst2	Bit 6	1	0	1
bst3	Bit 7	1	0	1
g0l0	Bit 8			
g0l1	Bit 9			
g0h0	Bit 10			
g0h1	Bit 11			
g1t4	Bit 12			
g1t3	Bit 13			
g2t4	Bit 14			
g2t3	Bit 15			
g3t4	Bit 16			
g3t3	Bit 17			
g4t4	Bit 18			
g4t3	Bit 19			
g5t4	Bit 20			
g5t3	Bit 21			
-	Bit 22			
-	Bit 23			
loop	Bit 24	0	0	0
exen	Bit 25	0	0	0
amx0	Bit 26	0	0	0
amx1	Bit 27	0	0	0
na	Bit 28	0	0	0
uta	Bit 29	1	0	1
todt	Bit 30	0	0	1
last	Bit 31	0	0	1
		WSS	**WSS+1**	**WSS+2**

Figure 15-43. Single Beat Write Access To Page Mode DRAM



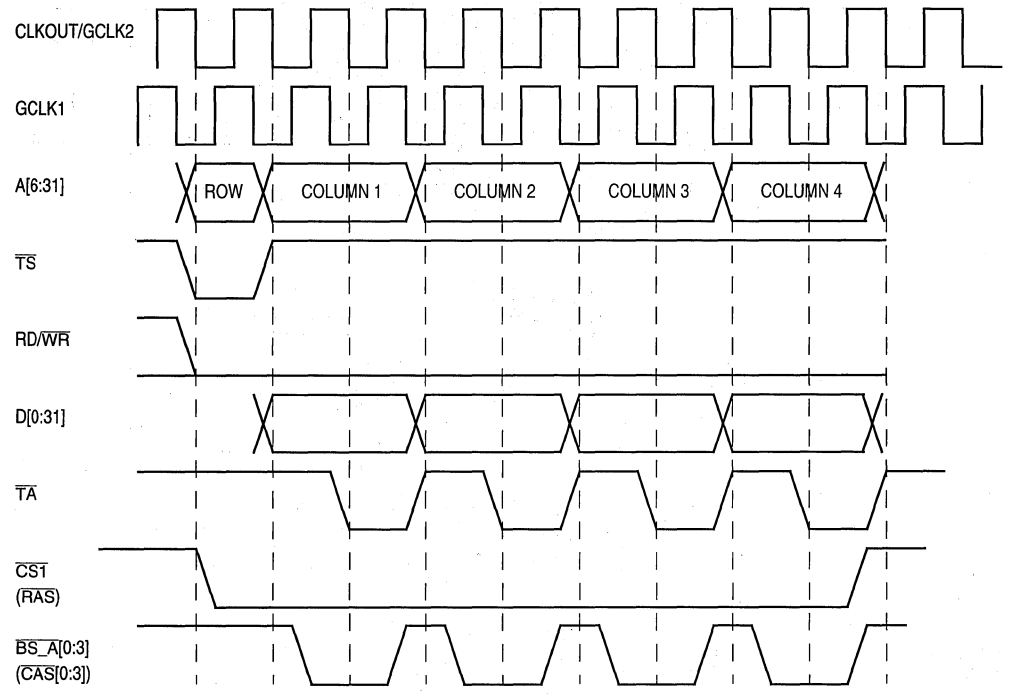
cst4	Bit 0	0	0	0	0	0	0	0	0	0
cst1	Bit 1	0	0	0	0	0	0	0	0	0
cst2	Bit 2	0	0	0	0	0	0	0	0	1
cst3	Bit 3	0	0	0	0	0	0	0	0	1
bst4	Bit 4	1	1	0	1	0	1	0	1	0
bst1	Bit 5	1	0	0	0	0	0	0	0	0
bst2	Bit 6	1	0	1	0	1	0	1	0	1
bst3	Bit 7	1	0	1	0	1	0	1	0	1
g0l0	Bit 8									
g0l1	Bit 9									
g0h0	Bit 10									
g0h1	Bit 11									
g1t4	Bit 12									
g1t3	Bit 13									
g2t4	Bit 14									
g2t3	Bit 15									
g3t4	Bit 16									
g3t3	Bit 17									
g4t4	Bit 18									
g4t3	Bit 19									
g5t4	Bit 20									
g5t3	Bit 21									
-	Bit 22									
-	Bit 23									
loop	Bit 24	0	0	0	0	0	0	0	0	0
exen	Bit 25	0	0	1	0	1	0	1	0	0
amx0	Bit 26	0	0	0	0	0	0	0	0	1
amx1	Bit 27	0	0	0	0	0	0	0	0	0
na	Bit 28	0	0	1	0	1	0	1	0	0
uta	Bit 29	1	0	1	0	1	0	1	0	1
todt	Bit 30	0	0	0	0	0	0	0	0	1
last	Bit 31	0	0	0	0	0	0	0	0	1
		RBS	**RBS+1**	**RBS+2**	**RBS+3**	**RBS+4**	**RBS+5**	**RBS+6**	**RBS+7**	**RBS+8**

Figure 15-44. Burst Read Access To Page Mode DRAM (No LOOP)



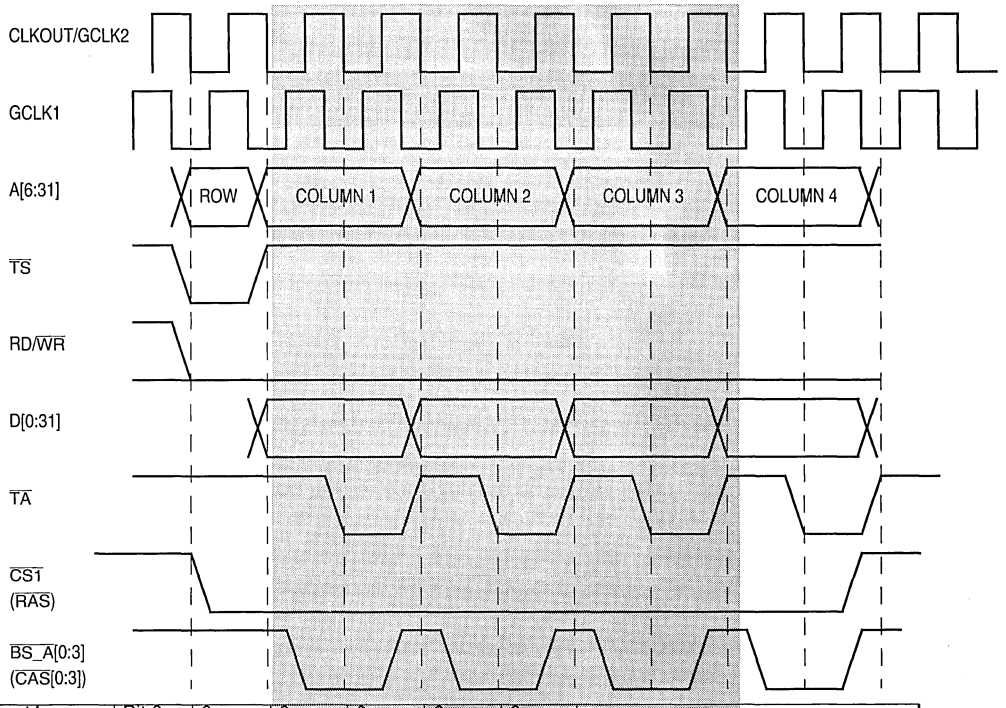
cst4	Bit 0	0	0	0	0	0
cst1	Bit 1	0	0	0	0	0
cst2	Bit 2	0	0	0	0	1
cst3	Bit 3	0	0	0	0	1
bst4	Bit 4	1	1	0	1	0
bst1	Bit 5	1	0	0	0	0
bst2	Bit 6	1	0	1	0	1
bst3	Bit 7	1	0	1	0	1
g0l0	Bit 8					
g0l1	Bit 9					
g0h0	Bit 10					
g0h1	Bit 11					
g1t4	Bit 12					
g1t3	Bit 13					
g2t4	Bit 14					
g2t3	Bit 15					
g3t4	Bit 16					
g3t3	Bit 17					
g4t4	Bit 18					
g4t3	Bit 19					
g5t4	Bit 20					
g5t3	Bit 21					
-	Bit 22					
-	Bit 23					
loop	Bit 24	0	1	1	0	0
exen	Bit 25	0	0	1	0	0
amx0	Bit 26	0	0	0	0	1
amx1	Bit 27	0	0	0	0	0
na	Bit 28	0	0	1	0	0
uta	Bit 29	1	0	1	0	1
todt	Bit 30	0	0	0	0	1
last	Bit 31	0	0	0	0	1
		RBS	RBS+1	RBS+2	RBS+3	RBS+4

Figure 15-45. Burst Read Access To Page Mode DRAM (LOOP)



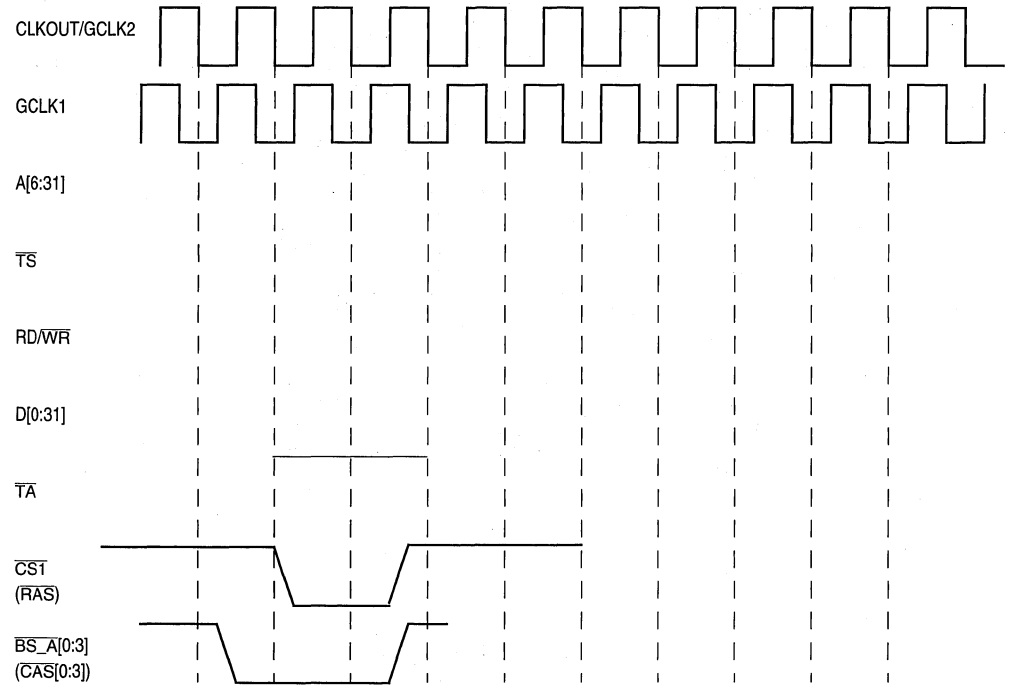
cst4	Bit 0	0	0	0	0	0	0	0	0	0
cst1	Bit 1	0	0	0	0	0	0	0	0	0
cst2	Bit 2	0	0	0	0	0	0	0	0	1
cst3	Bit 3	0	0	0	0	0	0	0	0	1
bst4	Bit 4	1	1	0	1	0	1	0	1	0
bst1	Bit 5	1	0	0	0	0	0	0	0	0
bst2	Bit 6	1	0	1	0	1	0	1	0	1
bst3	Bit 7	1	0	1	0	1	0	1	0	1
g0l0	Bit 8									
g0l1	Bit 9									
g0h0	Bit 10									
g0h1	Bit 11									
g1t4	Bit 12									
g1t3	Bit 13									
g2t4	Bit 14									
g2t3	Bit 15									
g3t4	Bit 16									
g3t3	Bit 17									
g4t4	Bit 18									
g4t3	Bit 19									
g5t4	Bit 20									
g5t3	Bit 21									
-	Bit 22									
-	Bit 23									
loop	Bit 24	0	0	0	0	0	0	0	0	0
exen	Bit 25	0	0	1	0	1	0	1	0	0
amx0	Bit 26	0	0	0	0	0	0	0	0	1
amx1	Bit 27	0	0	0	0	0	0	0	0	0
na	Bit 28	0	0	1	0	1	0	1	0	0
uta	Bit 29	1	0	1	0	1	0	1	0	1
todt	Bit 30	0	0	0	0	0	0	0	0	1
last	Bit 31	0	0	0	0	0	0	0	0	1
		WBS	WBS+1	WBS+2	WBS+3	WBS+4	WBS+5	WBS+6	WBS+7	WBS+8

Figure 15-46. Burst Write Access To Page Mode DRAM (No LOOP)



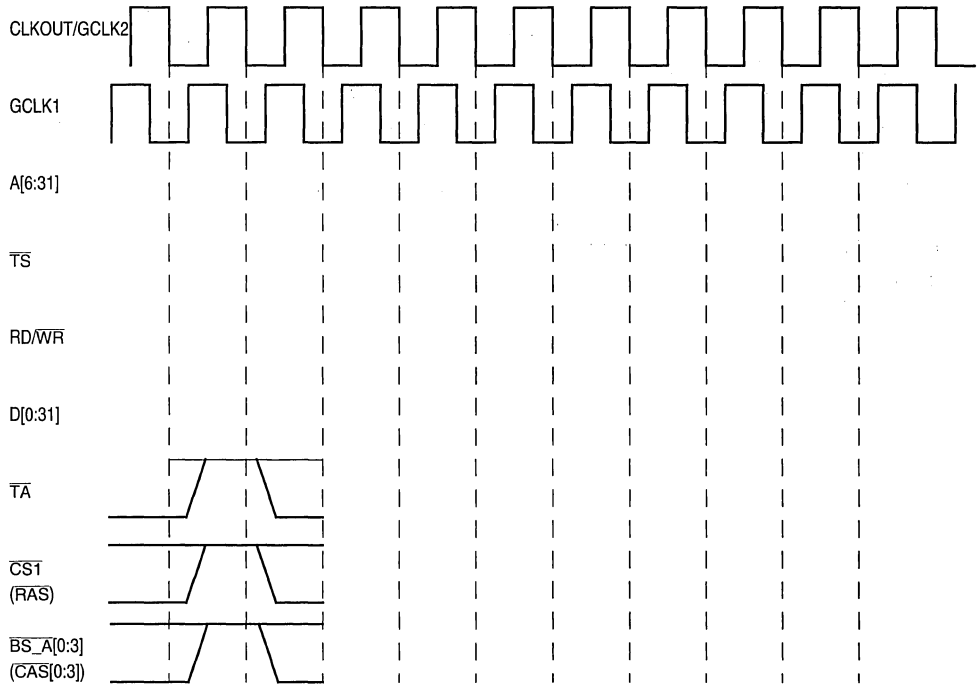
cst4	Bit 0	0	0	0	0	0	
cst1	Bit 1	0	0	0	0	0	
cst2	Bit 2	0	0	0	0	1	
cst3	Bit 3	0	0	0	0	1	
bst4	Bit 4	1	1	0	1	0	
bst1	Bit 5	1	0	0	0	0	
bst2	Bit 6	1	0	1	0	1	
bst3	Bit 7	1	0	1	0	1	
g0l0	Bit 8						
g0l1	Bit 9						
g0h0	Bit 10						
g0h1	Bit 11						
g1t4	Bit 12						
g1t3	Bit 13						
g2t4	Bit 14						
g2t3	Bit 15						
g3t4	Bit 16						
g3t3	Bit 17						
g4t4	Bit 18						
g4t3	Bit 19						
g5t4	Bit 20						
g5t3	Bit 21						
-	Bit 22						
-	Bit 23						
loop	Bit 24	0	1	1	0	0	
exen	Bit 25	0	0	1	0	0	
amx0	Bit 26	0	0	0	0	1	
amx1	Bit 27	0	0	0	0	0	
na	Bit 28	0	0	1	0	0	
uta	Bit 29	1	0	1	0	1	
todt	Bit 30	0	0	0	0	1	
last	Bit 31	0	0	0	0	1	
		WBS	WBS+1	WBS+2	WBS+3	WBS+4	

Figure 15-47. Burst Write Access To Page Mode DRAM (Loop)



cst4	Bit 0	1	0	0
cst1	Bit 1	1	0	0
cst2	Bit 2	1	0	1
cst3	Bit 3	1	0	1
bst4	Bit 4	1	0	0
bst1	Bit 5	0	0	0
bst2	Bit 6	0	0	1
bst3	Bit 7	0	0	1
g0l0	Bit 8			
g0l1	Bit 9			
g0h0	Bit 10			
g0h1	Bit 11			
g1t4	Bit 12			
g1t3	Bit 13			
g2t4	Bit 14			
g2t3	Bit 15			
g3t4	Bit 16			
g3t3	Bit 17			
g4t4	Bit 18			
g4t3	Bit 19			
g5t4	Bit 20			
g5t3	Bit 21			
-	Bit 22			
-	Bit 23			
loop	Bit 24	0	0	0
exen	Bit 25	0	0	0
amx0	Bit 26	0	0	1
amx1	Bit 27	0	0	0
na	Bit 28	0	0	0
uta	Bit 29	1	1	1
todt	Bit 30	0	0	1
last	Bit 31	0	0	1
		PTS	PTS+1	PTS+2

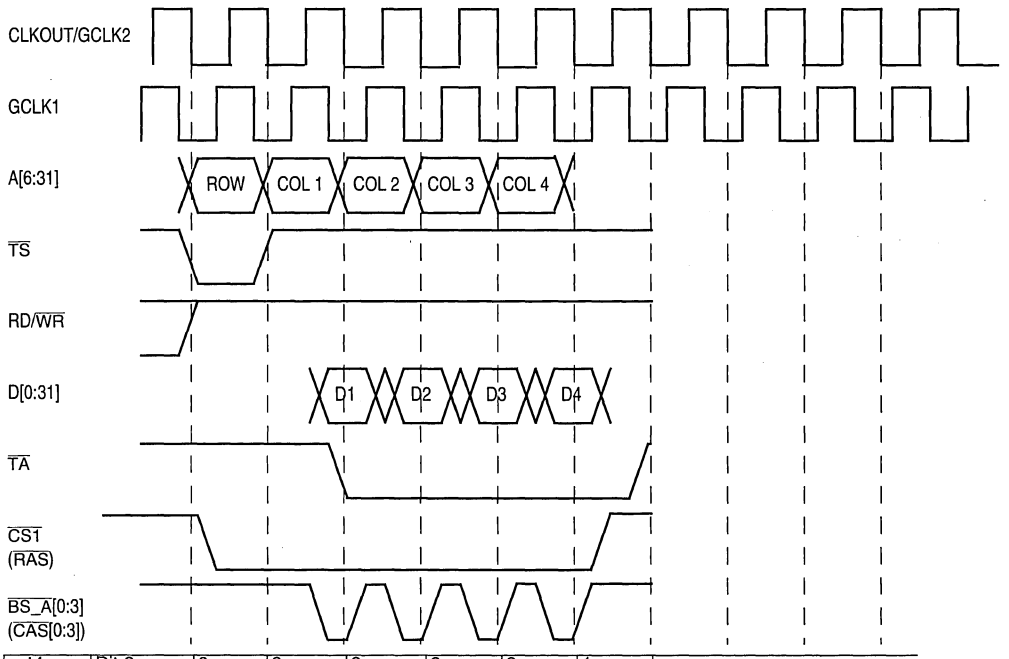
Figure 15-48. Refresh Cycle (CAS Before RAS) To Page Mode DRAM



cst4 cst1 cst2 cst3	Bit 0 Bit 1 Bit 2 Bit 3	1 1 1 1
bst4 bst1 bst2 bst3	Bit 4 Bit 5 Bit 6 Bit 7	1 1 1 1
g0l0 g0l1 g0h0 g0h1	Bit 8 Bit 9 Bit 10 Bit 11	
g1t4 g1t3	Bit 12 Bit 13	
g2t4 g2t3	Bit 14 Bit 15	
g3t4 g3t3	Bit 16 Bit 17	
g4t4 g4t3	Bit 18 Bit 19	
g5t4 g5t3	Bit 20 Bit 21	
- -	Bit 22 Bit 23	
loop exen	Bit 24 Bit 25	0 0
amx0 amx1 na uta todt last	Bit 26 Bit 27 Bit 28 Bit 29 Bit 30 Bit 31	0 0 0 1 1 1
		EXS

Figure 15-49. Exception Cycle

You can significantly increase the performance of a page read access when you set GPLA4DIS to 1 in the MAMR and ignore the $\overline{\text{GPL\_A4}}$ pin. The processor samples the data bus at the falling edge of GCLK1 when the $\overline{\text{TA}}$ signal is asserted. Figure 15-50 illustrates how to modify the burst read access to page mode DRAM (no loop) using this feature. During the four consecutive data beats, the $\overline{\text{TA}}$ signal in the figure is asserted to ensure a data transfer on every data clock. The figure also illustrates how the nine cycles of the burst read access shown in Figure 15-43 can be reduced to 6 clock cycles (for 32-bit port size memory). You can reduce the cycles by using faster DRAM or a slower system clock that meets the DRAM access time. When a 16-bit port size memory is connected, the reduction is from 17 to 10 cycles and when an 8-bit port size memory is connected, the reduction is from 33 to 18 cycles.



cst4	Bit 0	0	0	0	0	0	1	
cst1	Bit 1	0	0	0	0	0	1	
cst2	Bit 2	0	0	0	0	0	1	
cst3	Bit 3	0	0	0	0	0	1	
bst4	Bit 4	1	1	1	1	1	1	
bst1	Bit 5	1	0	0	0	0	1	
bst2	Bit 6	1	0	0	0	0	1	
bst3	Bit 7	1	0	0	0	0	1	
g0l0	Bit 8							
g0l1	Bit 9							
g0h0	Bit 10							
g0h1	Bit 11							
g1t4	Bit 12							
g1t3	Bit 13							
g2t4	Bit 14							
g2t3	Bit 15							
g3t4	Bit 16							
g3t3	Bit 17							
dlt3	Bit 18	1	1	1	1	1	1	
g4t3	Bit 19	0	0	0	0	0	0	
g5t4	Bit 20							
g5t3	Bit 21							
-	Bit 22							
-	Bit 23							
loop	Bit 24	0	0	0	0	0	0	
exen	Bit 25	0	0	0	0	0	0	
amx0	Bit 26	0	0	0	0	1	1	
amx1	Bit 27	0	0	0	0	0	0	
na	Bit 28	0	1	1	1	0	0	
uta	Bit 29	1	0	1	0	0	1	
todt	Bit 30	0	0	0	0	0	1	
last	Bit 31	0	0	0	0	0	1	
		RBS	**RBS+1**	**RBS+2**	**RBS+3**	**RBS+4**	**RBS+5**	

Figure 15-50. Optimized DRAM Burst Read Access

15.7.2 Page Mode Extended Data-Out DRAM Interface Example

The configuration for a 1M, 32-bit wide memory system using two 256K x 16-bit page mode extended data-out (EDO) DRAMs is illustrated in Figure 15-51. Also shown is the physical connection between UPMB and the EDO DRAMs. The $\overline{CS2}$ signal that is controlled by the base register is connected to both of the $\overline{RAS}$ signals. The $\overline{BS\_B}$[0:1] signals are mapped to the lower DRAM (D[0:15]) and the $\overline{BS\_B}$[2:3] signals are mapped to the upper DRAM (D[16:31]). For this connection, $\overline{GPL\_B1}$ is connected to the memory device $\overline{OE}$ pins. The refresh rate is calculated based on a 25MHz baud rate generator clock and the DRAM that requires a 512-cycle refresh every 8ms.

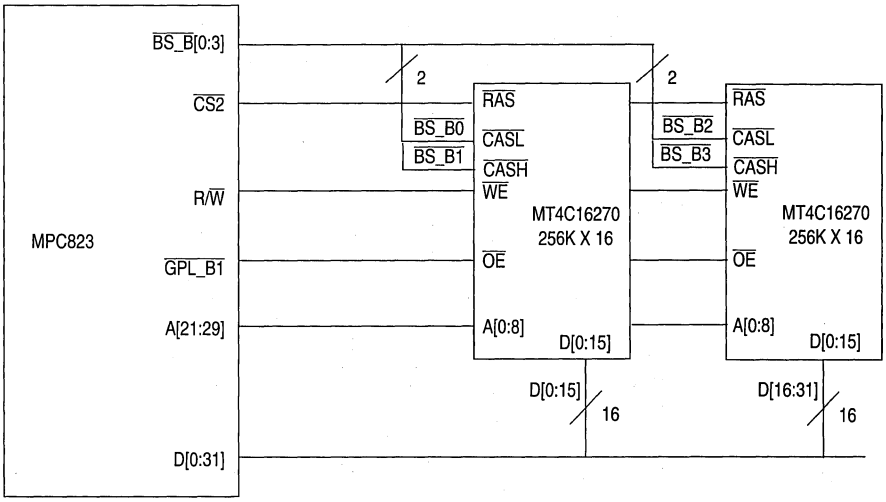


Figure 15-51. EDO DRAM Interface Connection

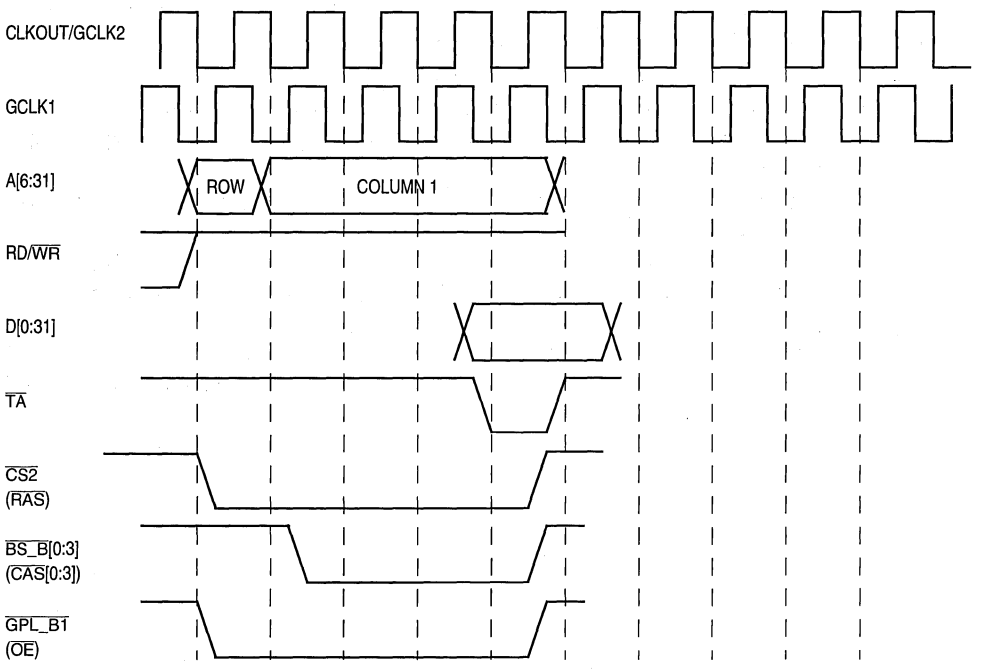
Follow these steps to configure a system for EDO DRAM:

1. Determine the system architecture, which includes the MPC823 and the memory system as shown in the example in Figure 15-52.
2. Use the blank worksheet in Figure 15-58 to draw the timing diagrams for all the memory cycles associated with your architecture. You can also use as a reference the various other timing diagrams in Figure 15-52 through Figure 15-57.
3. Translate the timing diagrams into RAM words for each type of memory access. The bottom half of the figures represent the RAM array contents that handle each of the possible cycles and each column represents a different word in the RAM array. A blank cell in the figures indicates a "don't care" bit, which is typically programmed to logic 1 to conserve power.

4. Define the UPMB (or UPMA) parameters that control the memory system in the following sequence. For additional details, see Table 15-11.
   - — Program the RAM array using the memory command register (MCR) and memory data register (MDR). The RAM word must be written into the MDR before you issue the **WRITE** command to the MCR. Repeat this step for all RAM word entries.
   - — Initialize the option and base registers of the specific bank according to the address mapping of the DRAM device you have chosen.
   - — Use the MS field of the option register to select the machine you have chosen to control the cycles. Notice that the SAM bit in the option register determines address multiplexing for the first clock cycle and subsequent cycles are controlled by the UPM RAM words. Also notice that the AMX field in the UPM RAM word controls the address multiplexing for the next clock cycle rather than the current cycle.
   - — Program the MBMR to select the number of columns and refresh timer parameters.

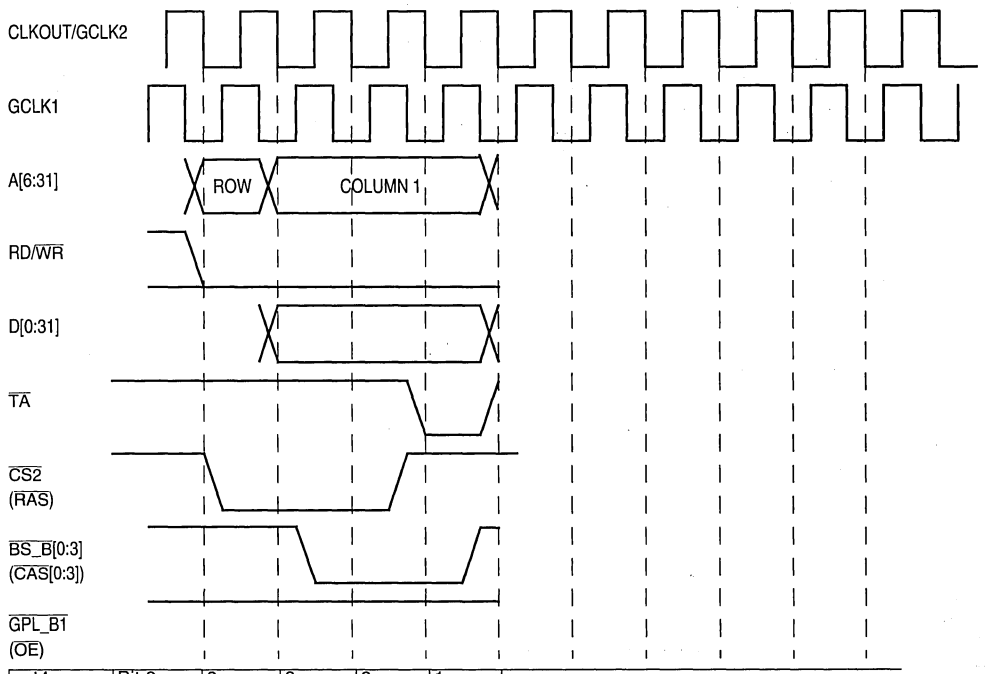
Table 15-11. UPMB Register Settings

FIELD	REGISTER	VALUE	COMMENTS
MS	BR2	10	Selects UPMB
PS	BR2	00	Selects 32-Bit Bus Width
WP	BR2	0	Allows Read and Write Accesses
PTP	MPTPR	00000010	Prescaler Divided by 32
PTB	MBMR	00001100	15.6µs at a 25MHz Clock
PTBE	MBMR	1	Enables Periodic Timer B
AMB	MBMR	001	Selects Nine Column Address Pins
DSB	MBMR	01	Selects Two Disable Timer Clock Cycles
GPLB4DIS	MBMR	0	Disables the UPWAITB Signal
RLFB	MBMR	0011	Selects Three Loop Iterations for Read
WLFB	MBMR	0011	Selects Three Loop Iterations for Write
SAM	OR2	1	Selects Column Address on First Cycle
BI	OR2	0	Supports Burst Accesses



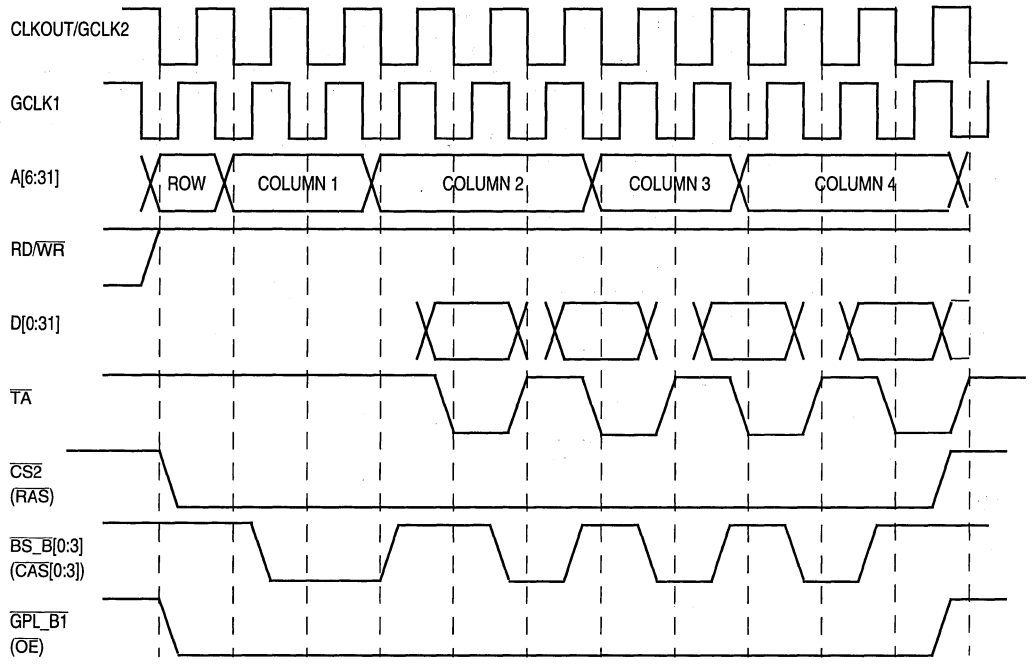
cst4	Bit 0	0	0	0	0	0	
cst1	Bit 1	0	0	0	0	0	
cst2	Bit 2	0	0	0	0	1	
cst3	Bit 3	0	0	0	0	1	
bst4	Bit 4	1	1	0	0	0	
bst1	Bit 5	1	0	0	0	0	
bst2	Bit 6	1	0	0	0	1	
bst3	Bit 7	1	0	0	0	1	
g0l0	Bit 8						
g0l1	Bit 9						
g0h0	Bit 10						
g0h1	Bit 11						
g1t4	Bit 12	0	0	0	0	0	
g1t3	Bit 13	0	0	0	0	1	
g2t4	Bit 14						
g2t3	Bit 15						
g3t4	Bit 16						
g3t3	Bit 17						
g4t4	Bit 18						
g4t3	Bit 19						
g5t4	Bit 20						
g5t3	Bit 21						
-	Bit 22						
-	Bit 23						
loop	Bit 24	0	0	0	0	0	
exen	Bit 25	0	0	0	0	0	
amx0	Bit 26	0	0	0	0	1	
amx1	Bit 27	0	0	0	0	0	
na	Bit 28	0	0	0	0	0	
uta	Bit 29	1	1	1	0	1	
todt	Bit 30	0	0	0	0	1	
last	Bit 31	0	0	0	0	1	
		RSS	**RSS+1**	**RSS+2**	**RSS+3**	**RSS+4**	

Figure 15-52. EDO DRAM Single Beat Read Access



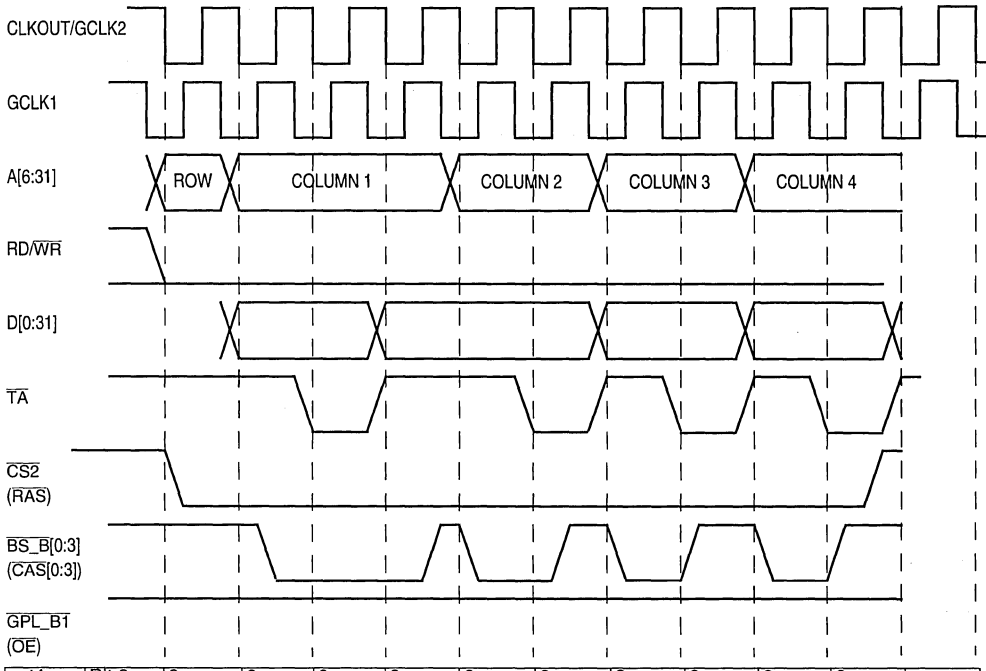
cst4	Bit 0	0	0	0	1
cst1	Bit 1	0	0	0	1
cst2	Bit 2	0	0	1	1
cst3	Bit 3	0	0	1	1
bst4	Bit 4	1	1	0	0
bst1	Bit 5	1	0	0	0
bst2	Bit 6	1	0	0	0
bst3	Bit 7	1	0	0	1
g0l0	Bit 8				
g0l1	Bit 9				
g0h0	Bit 10				
g0h1	Bit 11				
g1t4	Bit 12	1	1	1	1
g1t3	Bit 13	1	1	1	1
g2t4	Bit 14				
g2t3	Bit 15				
g3t4	Bit 16				
g3t3	Bit 17				
g4t4	Bit 18				
g4t3	Bit 19				
g5t4	Bit 20				
g5t3	Bit 21				
-	Bit 22				
-	Bit 23				
loop	Bit 24	0	0	0	0
exen	Bit 25	0	0	0	0
amx0	Bit 26	0	0	0	1
amx1	Bit 27	0	0	0	0
na	Bit 28	0	0	0	0
uta	Bit 29	1	1	0	1
todt	Bit 30	0	0	0	1
last	Bit 31	0	0	0	1
		WSS	**WSS+1**	**WSS+2**	**WSS+3**

Figure 15-53. EDO DRAM Single Beat Write Access



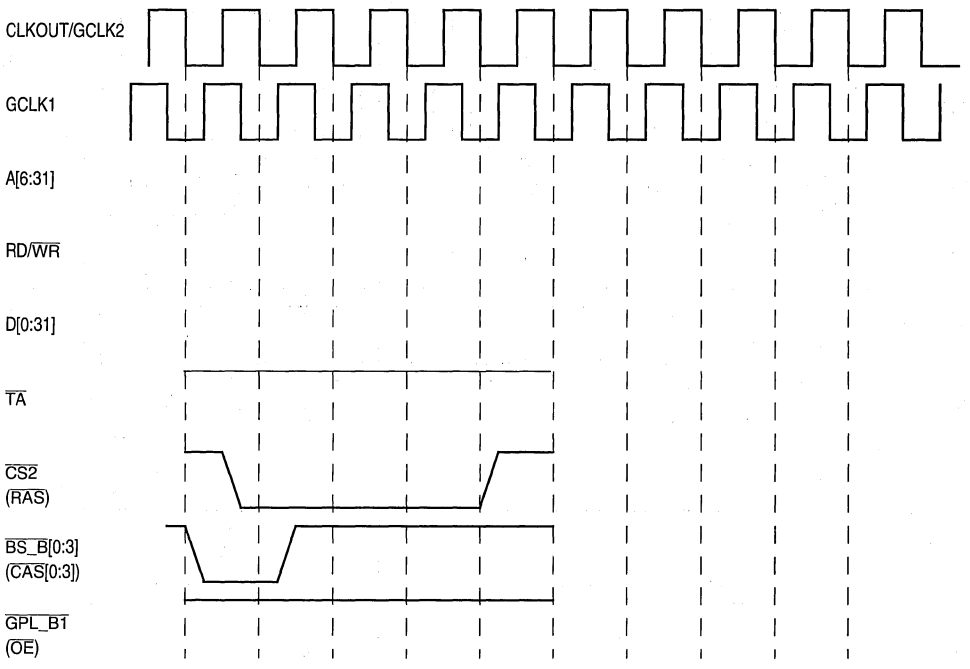
cst4	Bit 0	0	0	0	0	0	0	0	0	0	0	0
cst1	Bit 1	0	0	0	0	0	0	0	0	0	0	0
cst2	Bit 2	0	0	0	0	0	0	0	0	0	0	1
cst3	Bit 3	0	0	0	0	0	0	0	0	0	0	1
bst4	Bit 4	1	1	0	1	1	0	1	0	1	0	1
bst1	Bit 5	1	0	0	1	1	0	1	0	1	0	1
bst2	Bit 6	1	0	0	1	0	1	0	1	0	1	1
bst3	Bit 7	1	0	0	1	0	1	0	1	0	1	1
g0l0	Bit 8											
g0l1	Bit 9											
g0h0	Bit 10											
g0h1	Bit 11											
g1t4	Bit 12	0	0	0	0	0	0	0	0	0	0	0
g1t3	Bit 13	0	0	0	0	0	0	0	0	0	0	1
g2t4	Bit 14											
g2t3	Bit 15											
g3t4	Bit 16											
g3t3	Bit 17											
g4t4	Bit 18											
g4t3	Bit 19											
g5t4	Bit 20											
g5t3	Bit 21											
-	Bit 22											
-	Bit 23											
loop	Bit 24	0	0	0	0	0	0	0	0	0	0	0
exen	Bit 25	0	0	0	1	0	1	0	1	0	0	0
amx0	Bit 26	0	0	0	0	0	0	0	0	0	0	1
amx1	Bit 27	0	0	0	0	0	0	0	0	0	0	0
na	Bit 28	0	0	1	0	0	1	0	1	0	0	0
uta	Bit 29	1	1	1	0	1	0	1	0	1	0	1
todt	Bit 30	0	0	0	0	0	0	0	0	0	0	1
last	Bit 31	0	0	0	0	0	0	0	0	0	0	1
		RBS	**RBS+1**	**RBS+2**	**RBS+3**	**RBS+4**	**RBS+5**	**RBS+6**	**RBS+7**	**RBS+8**	**RBS+9**	**RBS+10**

Figure 15-54. EDO DRAM Burst Read Access



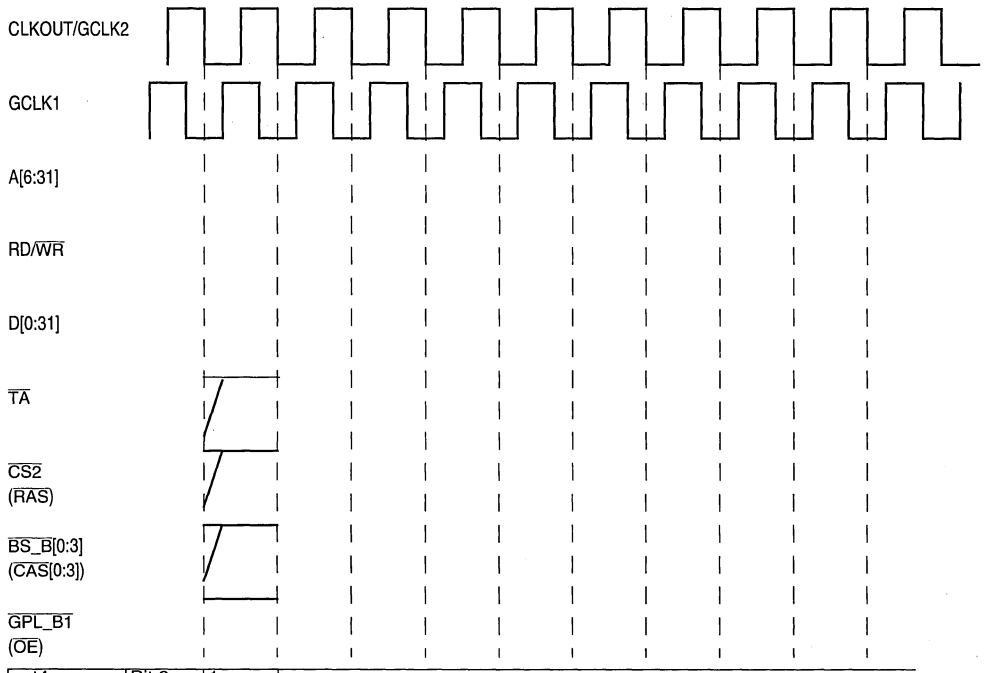
cst4	Bit 0	0	0	0	0	0	0	0	0	0	0
cst1	Bit 1	0	0	0	0	0	0	0	0	0	0
cst2	Bit 2	0	0	0	0	0	0	0	0	0	1
cst3	Bit 3	0	0	0	0	0	0	0	0	0	1
bst4	Bit 4	1	1	0	0	0	0	0	1	0	1
bst1	Bit 5	1	0	0	0	0	1	0	1	0	1
bst2	Bit 6	1	0	0	1	0	1	0	1	0	1
bst3	Bit 7	1	0	0	1	0	1	0	1	0	1
g0l0	Bit 8										
g0l1	Bit 9										
g0h0	Bit 10										
g0h1	Bit 11										
g1t4	Bit 12	1	1	1	1	1	1	1	1	1	1
g1t3	Bit 13	1	1	1	1	1	1	1	1	1	1
g2t4	Bit 14										
g2t3	Bit 15										
g3t4	Bit 16										
g3t3	Bit 17										
g4t4	Bit 18										
g4t3	Bit 19										
g5t4	Bit 20										
g5t3	Bit 21										
-	Bit 22										
-	Bit 23										
loop	Bit 24	0	0	0	0	0	0	0	0	0	0
exen	Bit 25	0	0	0	1	0	1	0	1	0	0
amx0	Bit 26	0	0	0	0	0	0	0	0	0	1
amx1	Bit 27	0	0	0	0	0	0	0	0	0	0
na	Bit 28	0	0	0	1	0	1	0	1	0	0
uta	Bit 29	1	0	1	1	0	1	0	1	0	1
todt	Bit 30	0	0	0	0	0	0	0	0	0	1
last	Bit 31	0	0	0	0	0	0	0	0	0	1
		WBS	**WBS+1**	**WBS+2**	**WBS+3**	**WBS+4**	**WBS+5**	**WBS+6**	**WBS+7**	**WBS+8**	**WBS+9**

Figure 15-55. EDO DRAM Burst Write Access



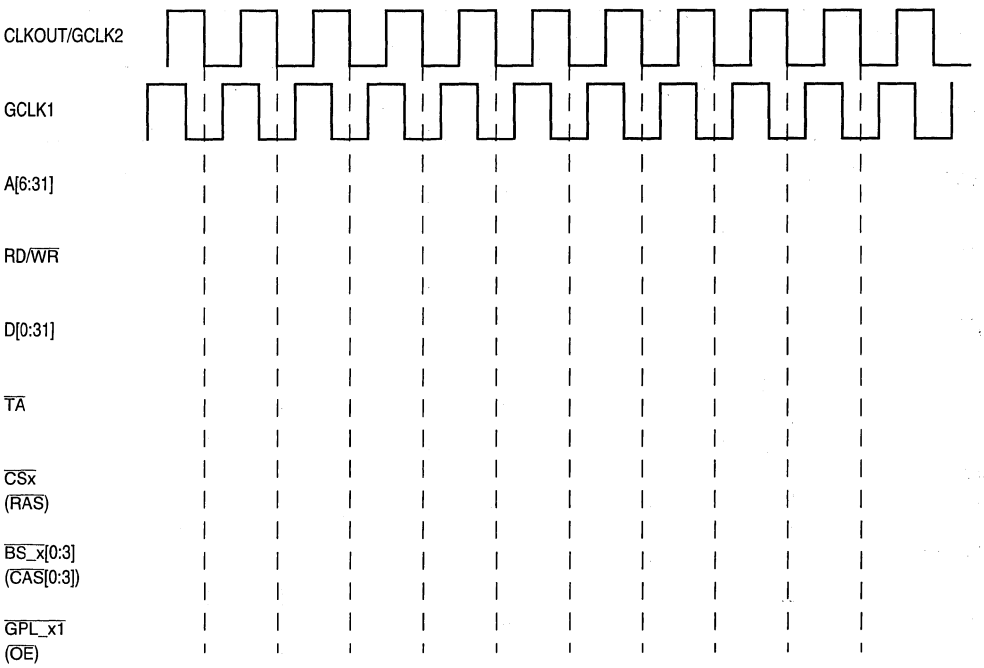
cst4	Bit 0	1	0	0	0	1	
cst1	Bit 1	1	0	0	0	1	
cst2	Bit 2	0	0	0	0	1	
cst3	Bit 3	0	0	0	0	1	
bst4	Bit 4	0	0	1	1	1	
bst1	Bit 5	0	1	1	1	1	
bst2	Bit 6	0	1	1	1	1	
bst3	Bit 7	0	1	1	1	1	
g0l0	Bit 8						
g0l1	Bit 9						
g0h0	Bit 10						
g0h1	Bit 11						
g1t4	Bit 12	1	1	1	1	1	
g1t3	Bit 13	1	1	1	1	1	
g2t4	Bit 14						
g2t3	Bit 15						
g3t4	Bit 16						
g3t3	Bit 17						
g4t4	Bit 18						
g4t3	Bit 19						
g5t4	Bit 20						
g5t3	Bit 21						
-	Bit 22						
-	Bit 23						
loop	Bit 24	0	0	0	0	0	
exen	Bit 25	0	0	0	0	0	
amx0	Bit 26	0	0	0	0	1	
amx1	Bit 27	0	0	0	0	0	
na	Bit 28	0	0	0	0	0	
uta	Bit 29	1	1	1	1	1	
todt	Bit 30	0	0	0	0	1	
last	Bit 31	0	0	0	0	1	
		PTS	**PTS+1**	**PTS+2**	**PTS+3**	**PTS+4**	

Figure 15-56. EDO DRAM Refresh Cycle (CAS Before RAS)



cst4 cst1 cst2 cst3	Bit 0 Bit 1 Bit 2 Bit 3	1 1 1 1
bst4 bst1 bst2 bst3	Bit 4 Bit 5 Bit 6 Bit 7	1 1 1 1
g0l0 g0l1 g0h0 g0h1	Bit 8 Bit 9 Bit 10 Bit 11	
g1t4 g1t3	Bit 12 Bit 13	1 1
g2t4 g2t3	Bit 14 Bit 15	
g3t4 g3t3	Bit 16 Bit 17	
g4t4 g4t3	Bit 18 Bit 19	
g5t4 g5t3	Bit 20 Bit 21	
- -	Bit 22 Bit 23	
loop exen	Bit 24 Bit 25	0 0
amx0 amx1 na uta todt last	Bit 26 Bit 27 Bit 28 Bit 29 Bit 30 Bit 31	0 0 0 1 1 1
		EXS

Figure 15-57. EDO DRAM Exception Cycle



cst4 cst1 cst2 cst3	Bit 0 Bit 1 Bit 2 Bit 3									
bst4 bst1 bst2 bst3	Bit 4 Bit 5 Bit 6 Bit 7									
g0l0 g0l1 g0h0 g0h1	Bit 8 Bit 9 Bit 10 Bit 11									
g1t4 g1t3	Bit 12 Bit 13									
g2t4 g2t3	Bit 14 Bit 15									
g3t4 g3t3	Bit 16 Bit 17									
g4t4 g4t3	Bit 18 Bit 19									
g5t4 g5t3	Bit 20 Bit 21									
- -	Bit 22 Bit 23									
loop exen	Bit 24 Bit 25									
amx0 amx1 na uta todt last	Bit 26 Bit 27 Bit 28 Bit 29 Bit 30 Bit 31									
		xxS	**xxS+1**	**xxS+2**	**xxS+3**	**xxS+4**	**xx+5**	**xxS+6**	**xxS+7**	**xxS+8**

Figure 15-58. Blank Worksheet for a UPM

SECTION 16
COMMUNICATION PROCESSOR MODULE

The MPC823 communication processor module (CPM) provides a flexible and integrated approach to communication-intensive environments. To reduce system frequency and save power, the communication processor module has its own independent RISC microcontroller that is optimized and tuned to handle serial communications. The communication processor module offloads the core in the following ways:

- By reducing the interrupt rate. The core is interrupted only upon frame reception or transmission, instead of on a per-character basis.
- By implementing some of the Layer-2 multiply-and-accumulate processing, which provides more CPU bandwidth for higher layer processing.
- By supporting multibuffer memory data structures that are convenient for software handling.

The communication processor module in the MPC823 and MPC821 are similar and are both derived from the communication processor module in the MC68360 QUICC. However, there are distinct differences, which are discussed in this section.

16.1 FEATURES

The following is a list of the communication processor module's main features. For quick reference purposes, the superscripted number following each item below is the page number where the feature is described.

- 32-Bit RISC Microcontroller[4]
- Flexibility of Four General-Purpose 16-Bit Timers or Two 32-Bit Timers[76]
- A Serial Interface with a Time-Slot Assigner [113]
- Four Independent Baud Rate Generators[151]
- A Full-Duplex Serial Communication Controller (SCC2)[157]
- A Universal Serial Bus Controller[342]
- Two Full-Duplex Serial Management Controllers (SMCs)[372]
- Serial Peripheral Interface (SPI) Support for Master or Slave[423]
- I^2C Bus Controller Support for Master or Slave[446]
- General-Purpose Parallel Interface Ports with Open-Drain Capability[467]
- Communication Processor Module Interrupt Controller with Flexible Priorities[487]

The following blocks and protocols contain primary programming functionalities that use parameter RAM tables as their communication interface.

- RISC microcontroller
  - ❑ Dual port RAM
  - ❑ RISC timer table
  - ❑ DSP functions
  - ❑ Independent DMA
- Serial communication controller
  - ❑ UART protocol
  - ❑ HDLC protocol
  - ❑ Appletalk/LocalTalk protocol
  - ❑ Asynchronous HDLC protocol
  - ❑ Infra-red protocols
  - ❑ Transparent protocol
  - ❑ Ethernet protocol
- Universal serial bus controller
- Serial management controllers
  - ❑ UART protocol
  - ❑ Transparent protocol
  - ❑ GCI protocol
- Serial peripheral interface
- I^2C bus controller
- General-purpose parallel interface ports

The block diagram of the communication processor module is illustrated in Figure 16-1.

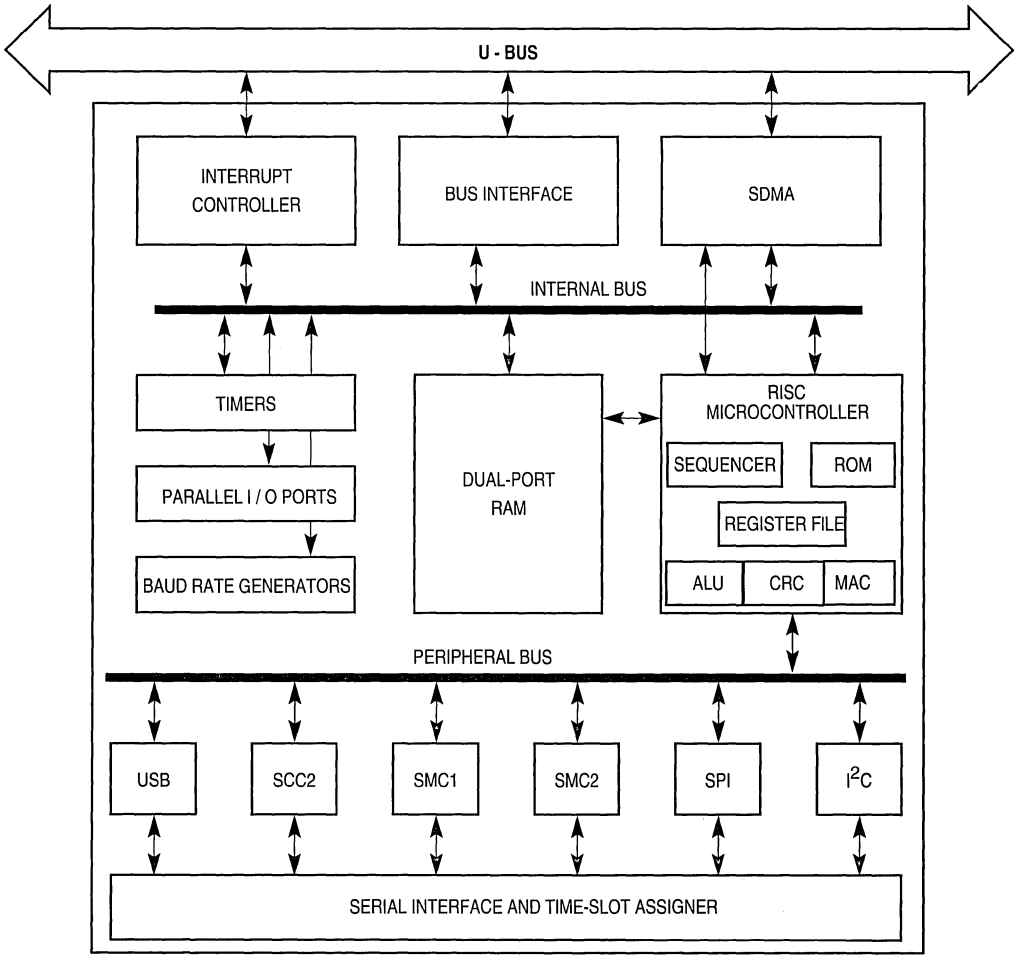


Figure 16-1. CPM Block Diagram

The MPC823 offers an extremely flexible set of communication capabilities. The remainder of this section discusses all the possible ways you can configure the communication processor module. Figure 16-2 illustrates a sample configuration for a personal digital assistant (PDA) application that supports various communication links and protocols.

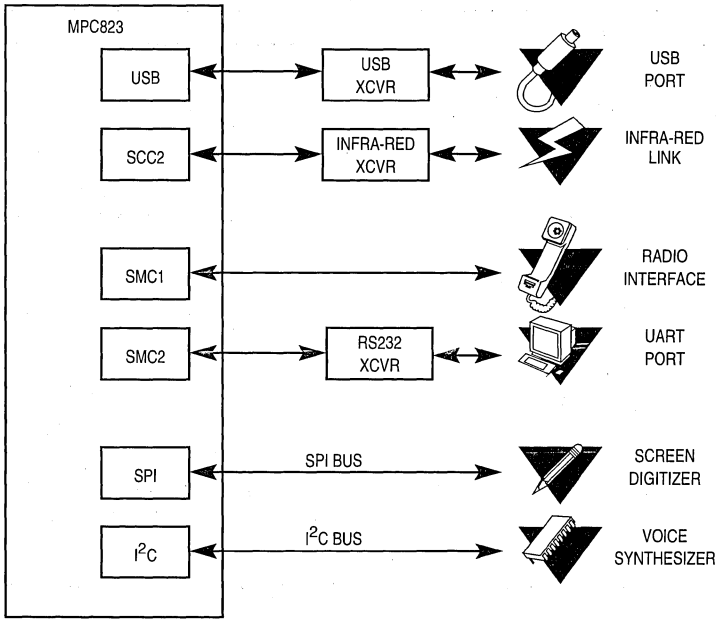


Figure 16-2. Example of a PDA Application

16.2 THE RISC MICROCONTROLLER

The 32-bit RISC microcontroller for the communication processor module resides on a separate bus from the core and, therefore, does not impact the core's performance. The microcontroller operates in conjunction with the serial channels and parallel port to implement the user-programmable protocols and manage the serial DMA (SDMA) channels that transfer data between the I/O channels and memory. The RISC microcontroller architecture and instruction set are optimized for data communication and data processing functions that are required by many wire-line and wireless communication standards. The microcontroller also contains a multiply and accumulate (MAC) unit composed of a 16x16-bit multiplier with two 40-bit accumulators that enable you to implement many DSP applications.

Basically, the microcontroller handles low-level arithmetic tasks and DMA control activities, which leaves the core free to handle the high-level activities. You could say it is the controller of the communication processor module. It manages IDMA channel operation and contains an internal timer that you can use to implement a maximum of 16 additional software timers.

16.2.1 RISC Microcontroller Features

The following is a list of the RISC microcontroller's main features.

- One System Clock Cycle Per Instruction
- Fixed-Length Instruction Object Code
- Code Is Executed from Internal ROM or Dual-Port RAM
- Automatically Switches to Low-Power Mode When Idle
- 32-Bit Data Path
- Optimized for Communication Processing
- Digital Signal Processing Capability Using MAC Arithmetic and Special Addressing Modes
- DMA Bursts Serial Data to External Memory

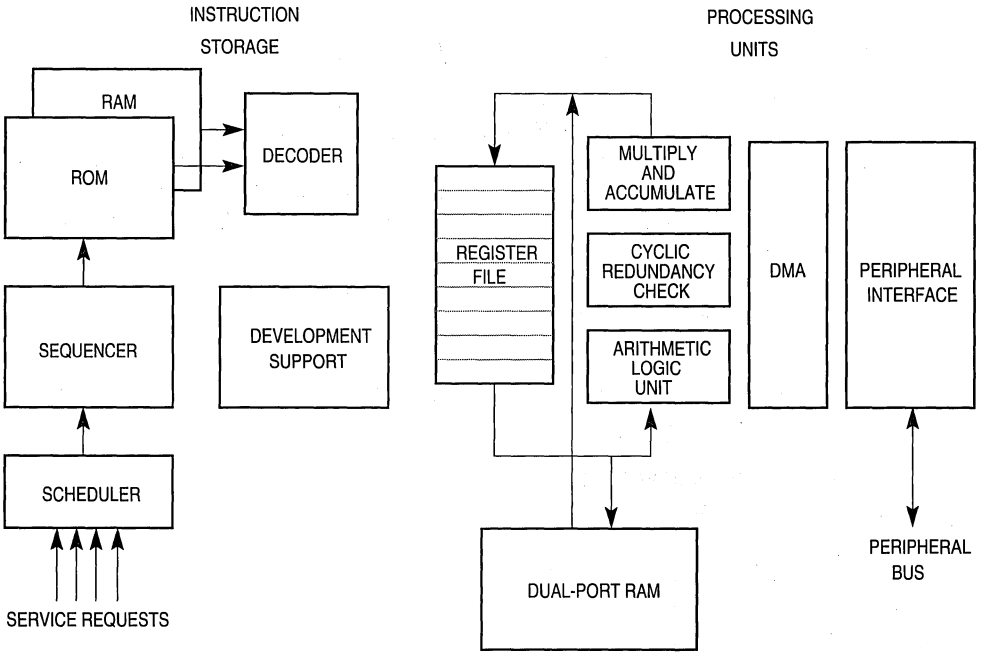


Figure 16-3. RISC Microcontroller Block Diagram

16.2.2 Communication Between the Microcontroller and Core

The RISC microcontroller communicates with the core in the following ways:

- By exchanging parameters using the 8K dual-port RAM. With simultaneous accesses, the microcontroller experiences a one-clock delay when accessing the dual-port RAM, but the host is never delayed.
- By executing special commands that are issued by the host via the CPM command register (CPCR). These commands should only be issued in special situations like exceptions or error recovery.
- By generating interrupts using the CPM interrupt controller.
- By allowing the core to configure the CPM via the RISC controller configuration register (RCCR)
- By allowing the core to read the CPM status and event registers at any time.

The core communicates with the communication processor module by configuring the RCCR.

16.2.3 Communication Between the Microcontroller and Peripherals

The RISC microcontroller uses the peripheral bus to communicate with all of its peripherals. The serial communication controller (SCC2) and universal serial bus (USB) have separate receive and transmit FIFOs. The SCC2 and USB FIFOs are 16 bytes each. However, the serial management controller, serial peripheral interface, and I^2C FIFO sizes are all double-buffered. The following prioritized list contains the processing order of the microcontroller from highest to lowest priority.

1. Reset in CPM command register or at reset
2. SDMA bus error
3. Commands issued to the command register, including DSP-related commands
4. IDMA DREQ1 (default setting)
5. IDMA DREQ2 (default setting)
6. USB Reception (RX)
7. USB Transmission (TX)
8. SCC2 RX
9. SCC2 TX
10. IDMA DREQ1 (option 2)
11. IDMA DREQ2 (option 2)
12. SMC1 RX
13. SMC1 TX
14. SMC2 RX
15. SMC2 TX

16. SPI RX
17. SPI TX
18. I^2C RX
19. I^2C TX
20. RISC timer tables
21. IDMA DREQ1 (option 3)
22. IDMA DREQ2 (option 3)

16.2.4 Executing Microcode From RAM or ROM

The microcontroller can execute microcode from a portion of 8K dual-port RAM. Depending on the size of your microcode, you can program the ERAM field in the RCCR to protect the first 512 bytes, 1,024 bytes, or 2,048 bytes of on-chip RAM to allow the microcontroller exclusive access. You can execute microcode from the dual-port RAM or on-chip ROM. This flexibility not only allows Motorola to add more protocols or enhancements to the MPC823, but it also allows you to obtain binary microcode. Refer to Table 16-1 for more information.

16.2.5 RISC Microcode Development Support Control Register

The RISC microcode development support control (RMDS) register determines which regions of the dual-port RAM can contain executable microcode. This register is used in conjunction with the ERAM field in the RCCR to determine the valid address space for executable microcode.

RMDS

BIT	0	1	2	3	4	5	6	7
FIELD	ERAM4K	RESERVED						
RESET	0	0						
R/W	R/W	R/W						
ADDR	(IMMR & 0xFFFF0000) + 0x9C7							

ERAM4K—Enable RAM Microcode (4K)

0 = Microcode program execution may be executed only from the first 2,048 bytes of the dual-port RAM.
1= Microcode is also executed from the 2,048 bytes of the second half of the dual-port RAM with a 512-byte extension.

Bits 1–7—Reserved

These bits are reserved and should be set to 0.

16.2.6 RISC Controller Configuration Register

The 16-bit, memory-mapped, read/write RISC controller configuration register (RCCR) configures the microcontroller to run microcode from ROM or RAM and controls the RISC internal timer. This register is initialized to zero at reset.

RCCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	TIME	RES	TIMEP						DR2M	DR1M	DRQP		EIE	SCD	ERAM	
RESET	0	0	0						0	0	0		0	0	0	
R/W	R/W	R/W	R/W						R/W	R/W	R/W		R/W	R/W	R/W	
ADDR	(IMMR & 0xFFF0000) + 0x9C4F															

TIME—Timer Enable

This bit controls whether the microcontroller's internal timer can send a tick to the microcontroller based on the value programmed into the TIMEP bit.

0 = Stop RISC timer table scanning.
1 = Start RISC timer table scanning.

Bit 1—Reserved

This bit is reserved and should be set to 0.

TIMEP—Timer Period

This field controls the microcontroller's timer tick. The RISC timer tables are scanned on each timer tick and the input to the timer tick generator is the general system clock divided by 1,024. The formula is (TIMEP + 1) × 1,024 = (general system clock period). Thus, a value of 0 stored in these bits gives a timer tick of 1 × (1,024) = 1,024 general system clocks and a value of 63 (decimal) gives a timer tick of 64 × (1,024) = 65,536 general system clocks.

DR2M—IDMA Request 2 Mode

This bit controls the IDMA request 1 ($\overline{\text{DREQ2}}$) sensitivity mode.

0 = $\overline{\text{DREQ2}}$ is edge-sensitive.
1 = $\overline{\text{DREQ2}}$ is level-sensitive.

DR1M—IDMA Request 1 Mode

This bit controls the IDMA request 0 ($\overline{\text{DREQ1}}$) sensitivity mode.

0 = $\overline{\text{DREQ1}}$ is edge-sensitive.
1 = $\overline{\text{DREQ1}}$ is level-sensitive.

DRQP—IDMA Emulation Request Priority

This field controls the priority of the external requests signals that relate to the serial channels. Refer to **Section 16.2.3 Communication Between the Microcontroller and Peripherals** for more information.

00 = IDMA requests have priority over the serial communication controller and USB (default).
01 = IDMA requests have priority immediatley following the serial communication controller (option 2).
10 = IDMA requests have the lowest priority (option 3).
11 = Reserved.

EIE—External Interrupt Enable

Configure this bit as instructed in the download process of a Motorola-supplied RAM microcode package. This bit is also used by IDMA channel 1 to enable single-buffer mode, as described in **Section 16.6.3.11.4 Single-Buffer Burst Fly-By Mode**.

0 = $\overline{\text{DREQ1}}$ pin cannot interrupt the microcontroller.
1 = $\overline{\text{DREQ1}}$ pin can interrupt the microcontroller.

SCD—Scheduler Configuration

Configure this bit as instructed in the download process of a Motorola-supplied RAM microcode package.

0 = Normal operation.
1 = Alternate configuration of the scheduler.

ERAM—Enable RAM Microcode

Configure this field as instructed in the download process of a Motorola-supplied RAM microcode package. This field is used in conjunction with the ERAM 4K bit in the RMDS register to configure the RAM microcode space.

Table 16-1. RAM Microcode Configurations

ERAM (RCCR)	BIT 0 IN RMDS	MICROCODE ADDRESSES
01	0	2000-21ff and 2f00-2fff
10	0	2000-23ff and 2f00-2fff
11	0	2000-27ff and 2e00-2fff
01	1	2000-21ff, 2f00-2fff, 3000-37ff, and 3a00-3bff
10	1	2000-23ff, 2f00-2fff, 3000-37ff, and 3a00-3bff
11	1	2000-27ff, 2e00-2fff, 3000-37ff, and 3a00-3bff

16.2.7 RISC Microcontroller Commands

To initialize the serial channel or DMA, you can issue a command to the CPM command register. The command you issue will ask the communication processor module to perform further device-specific functions based on the information in the device's parameter RAM.

16.2.7.1 CPM COMMAND REGISTER. The core sets the FLG bit in the 16-bit, memory-mapped, read/write CPM command register (CPCR) when it issues a command and the communication processor module clears the FLG bit when the command is completed. The core is now ready for the next command. Subsequent commands to the CPCR can only be given when the FLG bit is clear. When issuing the software reset command, the core should also set the FLG bit.

CPCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RST	RESERVED			OPCODE				CH_NUM				RESERVED			FLG
RESET	0	0			0				0				0			0
R/W	R/W	R/W			R/W				R/W				R/W			R/W
ADDR	(IMMR & 0xFFFF0000) + 0x9C0															

RST—Software Reset Command

This bit is set by the core and cleared by the communication processor module and when this command is executed, the RST and FLG bits are cleared within two general system clocks. The RISC reset routine is approximately 60 clocks long, but you can start initializing the communication processor module immediately after this command is issued. RST is useful when the core wants to reset the registers and parameters for all the channels as well as the RISC microprocessor and timer tables. However, this bit does not affect the serial interface or parallel I/O registers.

0 = No reset is issued.
1 = Reset is issued.

Bits 1–3—Reserved

These bits are reserved and should be set to 0.

OPCODE—Operation Code

This field is used in conjunction with the CH_NUM field to define a command sent to the CPM. It issues a variety of commands, which are described in Table 16-2. For the same operation code, the results may be different, depending on the channel number you select. For example, if your operation code is 0101 (**GRACEFUL STOP TX**) and your channel number is set to 0100 (SCC2), then the operation will gracefully stop the transmit on SCC2. If your channel number is set to 0001 (IDMA1), then the operation will gracefully stop the transmit on IDMA1.

CH_NUM—Channel Number

This field is set by the core to define the peripheral I/O channel that the command is applied to. Some peripherals share channel number encodings if their commands are mutually exclusive. See Table 16-2 for more information.

Bits 12–14—Reserved

These bits are reserved and should be set to 0.

FLG—Command Semaphore Flag

The bit is set by the core and cleared by the communication processor module.

0 = The communication processor module is ready to receive a new command.
1 = The CPCR contains a command that the communication processor module is currently processing. The communication processor module clears this bit when the command finishes executing or after reset.

16.2.7.2 COMMAND DEFINITIONS. The RISC microcontroller requires an opcode and a channel number to determine which command to issue. These opcodes and their definitions are described below. The opcodes and channel numbers that appear in Table 16-2 are actually the commands to be issued in the CPCR.

Table 16-2. RISC Microcontroller Commands

OPCODE	CHANNEL NUMBER										
	SCC2 (0100)	USB (0000)	SMC1 (1001) OR SMC2 (1101) (UART/ TRANS)	SMC1 (1001) OR SMC2 (1101) (GCI)	SPI (0101)	I^2C (0001)	IDMA1 (0001)	IDMA2 (0101)	DSP1 RX (1001)	DSP2 TX (1101)	TIMER (0101)
0000	INIT RX AND TX PARAMS	INIT RX AND TX PARAMS	INIT RX AND TX PARAMS	INIT RX AND TX PARAMS	INIT RX AND TX PARAMS	INIT RX AND TX PARAMS	—	—	—	—	—
0001	INIT RX PARAMS	INIT RX PARAMS	INIT RX PARAMS	—	INIT RX PARAMS	INIT RX PARAMS	—	—	—	—	—
0010	INIT TX PARAMS	INIT TX PARAMS	INIT TX PARAMS	—	INIT TX PARAMS	INIT TX PARAMS	—	—	—	—	—
0011	ENTER HUNT MODE	ENTER HUNT MODE	ENTER HUNT MODE	—	—	—	—	—	—	—	—
0100	STOP TX	STOP TX	STOP TX	—	—	—	—	—	—	—	—
0101	GRACEFUL STOP TX	GRACEFUL STOP TX	—	—	—	—	INIT IDMA	INIT IDMA	—	—	—
0110	RESTART TX	RESTART TX	RESTART TX	—	—	—	—	—	—	—	—
0111	CLOSE RX BD	CLOSE RX BD	CLOSE RX BD	—	CLOSE RX BD	CLOSE RX BD	—	—	—	—	—
1000	SET GROUP ADDRESS	—	—	—	—	—	—	—	—	—	SET TIMER
1001	—	—	—	GCI TIMEOUT	—	—	—	—	—	—	—
1010			—	GCI ABORT REQUEST	—	—	—	—	—	—	—
1011	—	—	—	—	—	—	STOP IDMA	STOP IDMA	—	—	—
1100	—	—	—	—	—	—	—	—	START DSP	START DSP	—
1101	—	—	—	—	—	—	ARM IDMA	ARM IDMA	INIT DSP	INIT DSP	—
1110	—	—	—	—	—	—	—	—	—	—	—
1111	—	USB Command	—	—	—	—	—	—	—	—	—

NOTE: — = Reserved.

The RISC microcontroller commands consist of the following:

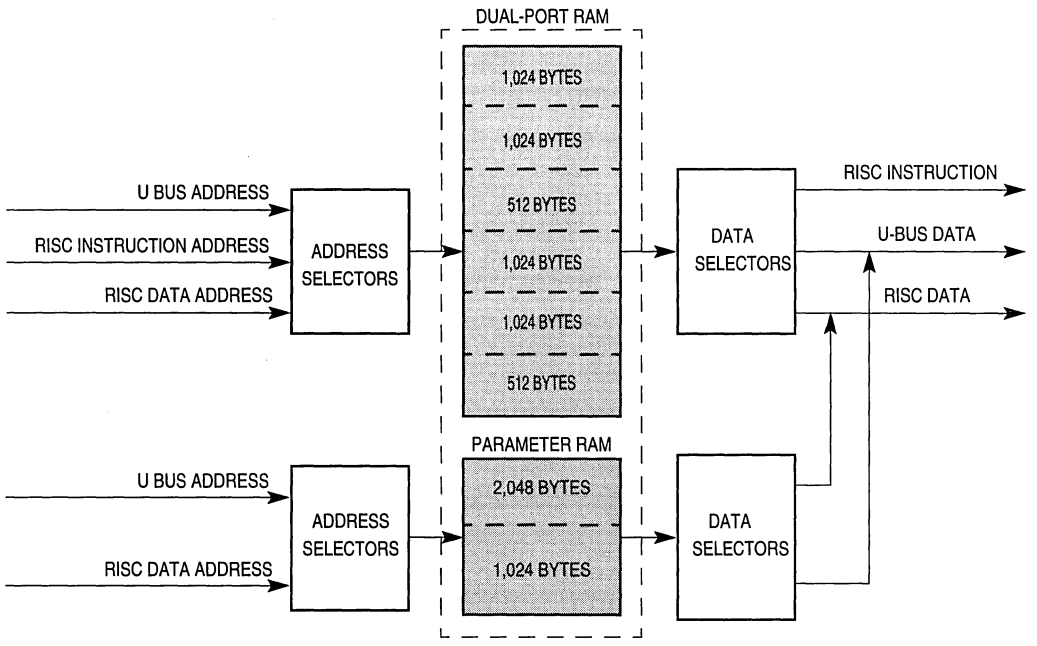
- **INIT TX AND RX PARAMS**—The initialize transmit and receive parameter command initializes the transmit and receive parameters in the parameter RAM to the values that they had when the communication processor module was last reset. This command is especially useful when switching protocols on a serial channel.
- **INIT RX PARAMETERS**—The initialize receive parameters command initializes the receive parameters of the serial channel.
- **INIT TX PARAMETERS**—The initialize transmit parameters command initializes the transmit parameters of the serial channel.
- **ENTER HUNT MODE**—The enter hunt mode command causes the receiver to stop receiving and start looking for a new frame. The exact operation of this command depends on the protocol that is used.
- **STOP TX**—The stop transmission command stops transmitting from this channel as soon as the transmit FIFO has been emptied. It should only be used when transmissions need to be stopped as quickly as possible. Transmission continues when the **RESTART TX** command is issued.
- **GRACEFUL STOP TX**—The graceful stop transmission command stops transmitting from this channel as soon as the current frame has been fully transmitted from the transmit FIFO. Transmission continues when the **RESTART TX** command is issued and the R bit is set in the next transmit buffer descriptor.
- **RESTART TX**—Once the **STOP TX** command has been issued, the restart transmission command is used to start transmitting again at the current buffer descriptor.
- **CLOSE RX BD**—The close receive buffer descriptor command causes the receiver to close the current receive buffer descriptor, which makes the receive buffer available for you to manipulate. The next available buffer descriptor is used to continue reception. You can use this command to access the data buffer and you won't have to wait until the serial communication controller fills it.
- **INIT IDMA**—The initialize IDMA command initializes the IDMA internal state to the value it had at system reset. It is only required when the IDMA autobuffer or buffer chaining modes are used.
- **ARM IDMA**—The arm IDMA command causes the IDMA to open the next buffer descriptor in the table. It can be used to reduce the latency of servicing the first IDMA request.
- **SET TIMER**—The set timer command is used to activate, deactivate, or reconfigure the 16 timers of the RISC timer table.
- **SET GROUP ADDRESS**—The set group address command sets a hash table bit for the Ethernet logical group address recognition function.
- **GCI ABORT REQUEST**—The GCI abort request command causes the MPC823 receiver to send an abort request on the A bit of the GCI bus.

- **GCI TIMEOUT**—The GCI timeout command causes the MPC823 transmitter to send an abort request on the E bit of the GCI bus.
- **RESET BCS**—The reset block check sequence command is used in BISYNC mode to reset the block check sequence calculation.
- **USB**—The USB commands have the same opcode. See **Section 16.10 Universal Serial Bus Controller** for a more detailed description.

16.2.7.2.1 CPM Command Register Examples. For example, to perform a complete reset of the communication processor module, you should write 0x8001 to the CPCR, which also sets the RST and FLG bits. After you issue this command, the CPCR returns a value of 0x0000 after two clocks. To execute an **ENTER HUNT MODE** command to the SCC2, write 0x0341 to the CPCR. While this command is executing, the CPCR returns a 0x0341 value and once it is finished it returns a 0x0340 value, which clears the FLG bit.

Note: The worst-case command execution latency is 120 clocks and the typical command execution latency is approximately 40 clocks.

16.2.7.3 DUAL-PORT RAM. The communication processor module has 8,192 bytes of static RAM that is configured as dual-port memory. Revision 0 of the MPC823 silicon has a 5K dual-port RAM. A block diagram of the dual-port RAM is illustrated in Figure 16-4.



NOTE: The shaded area indicates the area that is implemented on the silicon.

Figure 16-4. Dual-Port RAM Block Diagram

The dual-port RAM can either be accessed by the RISC microcontroller or one of two bus masters—the core or the serial DMA channel. When the dual-port RAM is accessed by one of these, it is accessed in two clocks. However, when it is accessed by the microcontroller, it is accessed in one clock. When simultaneous accesses occur with at least one write operation, the microcontroller is delayed by one clock.

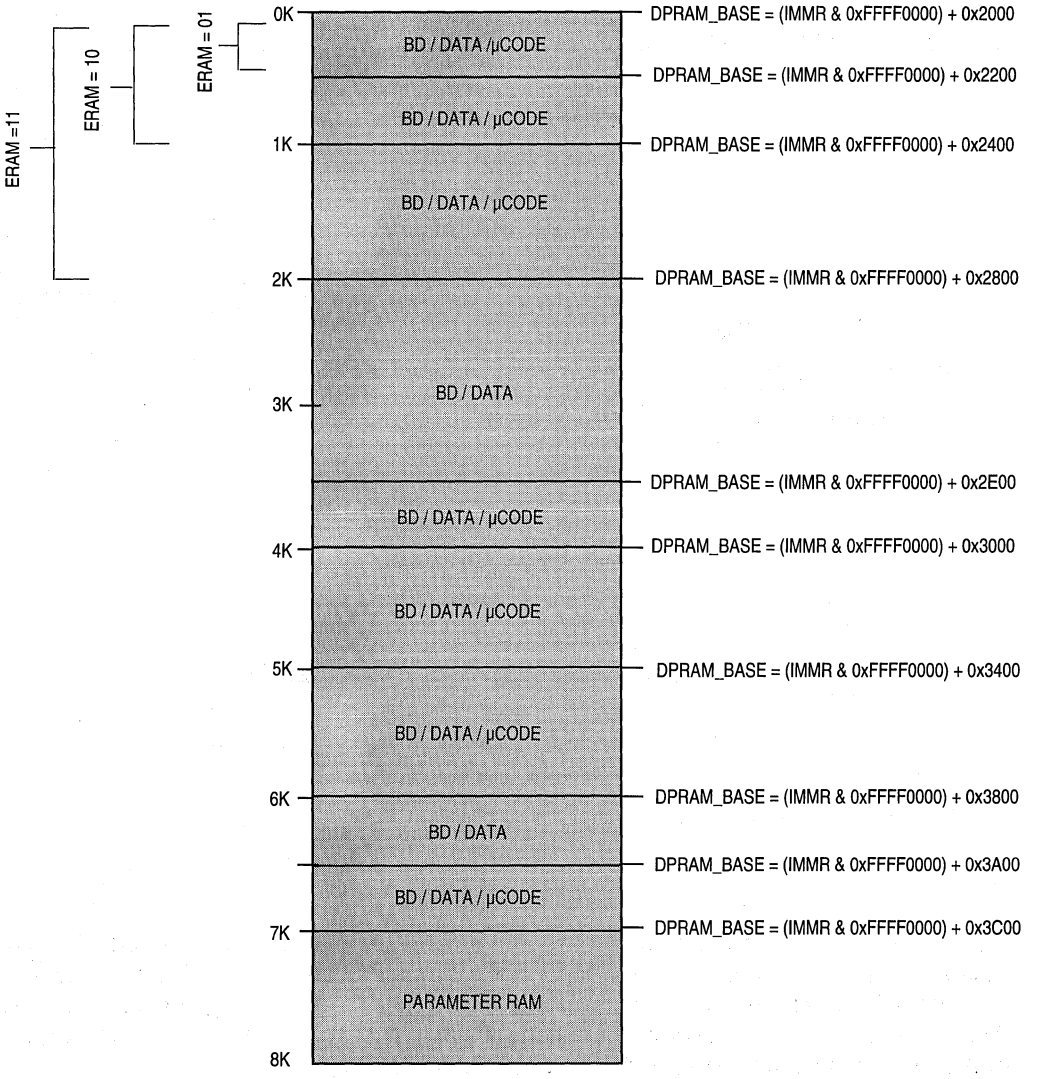


Figure 16-5. Dual-Port RAM Memory Map

When the dual-port RAM is accessed by the core or SDMA channel, the data and address are passed to and from the U-bus. The microcontroller has access to the entire dual-port RAM for data fetches and portions of the system RAM for microcode instruction fetches.

The dual-port RAM is used to complete the following tasks. Any two of them can occur simultaneously.

- To store the parameters associated with the USB, SCC2, SMCs, SPI, I²C, and IDMAs in the 1,024-byte parameter RAM.
- To store the buffer descriptors that describe where data is to be received and transmitted.
- To store the data from the serial channels. This is optional because data can also be stored in external memory.
- To store the RAM microcode for the RISC microcontroller. This feature allows Motorola to add protocols in the future.
- To use as an additional scratchpad RAM space for your program.

Only the content of the parameter and microcode RAM options require the use of fixed addresses. The buffer descriptors, buffer data, and scratchpad RAM can be located in the internal system RAM or in any unused parameter RAM. For instance, the area that is available when a serial channel or sub-block is not being used. When a microcode from RAM is executed, certain portions of the system RAM are no longer available. There are three possible configurations for microcode area sizes— first 512-byte block with a 256-byte extension (RCCR ERAM=01), first 1,024-byte block with 256-byte extension (RCCR ERAM=10), or first 2,048-byte block with 512-byte extension (RCCR ERAM=11). The remainder of the first 4,096 bytes are available as system RAM. See Table 16-1 for details.

16.2.7.3.1 Buffer Descriptors. The universal serial bus, serial communication controller, serial management controllers, serial peripheral interface, and I²C always use buffer descriptors to control data buffers. The table below shows that their buffer descriptor formats are all the same. If the IDMA channel is used in buffer chaining or autobuffer mode, it also uses buffer descriptors.

	0 15
OFFSET + 0	STATUS AND CONTROL
OFFSET + 2	DATA LENGTH
OFFSET + 4	HIGH-ORDER DATA BUFFER POINTER
OFFSET + 6	LOW-ORDER DATA BUFFER POINTER

16.2.7.3.2 Parameter RAM. The communication processor module maintains a section of dual-port RAM called the parameter RAM. It contains many parameters for a universal serial bus, serial communication controller, serial management controller, serial peripheral interface, I²C controller, and IDMA channel operation. The parameter RAM structure is summarized in Table 16-3.

A definition of the parameter RAM is contained in each protocol subsection that describes the device using a parameter RAM. For example, in some locations the Ethernet parameter RAM memory map is defined in the same way that the HDLC specific parameter RAM is defined.

Table 16-3. Parameter RAM Memory Map

	PAGE	ADDRESSES	PERIPHERAL
IMMR + 0x3C00	1	DPRAM_Base+ 0x1c00	USB
		DPRAM_Base+ 0x1c7f	
		DPRAM_Base+ 0x1c80	I^2C
		DPRAM_Base+ 0x1caf	
		DPRAM_Base+ 0x1cb0	MISC
		DPRAM_Base+ 0x1cbf	
		DPRAM_Base+ 0x1cc0	IDMA1
		DPRAM_Base+ 0x1cff	
IMMR + 0x3D00	2	DPRAM_Base+ 0x1d00	SCC2
		DPRAM_Base+ 0x1d7f*	
		DPRAM_Base+ 0x1d80	SPI
		DPRAM_Base+ 0x1daf	
		DPRAM_Base+ 0x1db0	Timers
		DPRAM_Base+ 0x1dbf	
		DPRAM_Base+ 0x1dc0	IDMA2
		DPRAM_Base+ 0x1dff	
IMMR + 0x3E00	3	DPRAM_Base+ 0x1e00	Reserved
		DPRAM_Base+ 0x1e7f	
		DPRAM_Base+ 0x1e80	SMC1
		DPRAM_Base+ 0x1ebf	
		DPRAM_Base+ 0x1ec0	DSP1
		DPRAM_Base+ 0x1eff	
IMMR + 0x3F00	4	DPRAM_Base+ 0x1f00	Reserved
		DPRAM_Base+ 0x1f7f	
		DPRAM_Base+ 0x1f80	SMC2
		DPRAM_Base+ 0x1fbf	
		DPRAM_Base+ 0x1fc0	DSP2
		DPRAM_Base+ 0x1fff	

NOTE: DPRAM_Base = (IMMR & 0xFFFF0000) + 0x2000.
* 0x1da3 for Ethernet.

16.2.7.4 THE RISC TIMER TABLES. The RISC microcontroller can have a maximum of 16 timers that are separate and distinct from the four general-purpose timers and baud rate generators of the communication processor module. These timers are ideal for protocols that do not require extreme precision, but do need to free the host CPU from scanning the software's timer tables. These timers are clocked from an internal timer that only the microcontroller can access. Each pair of timers can be configured as pulse width modulation (PWM) channels. The output of the channel is driven on one of the port B pins and a maximum of six PWM channels are supported. The following list summarizes the main features of the RISC timer tables.

- Supports a maximum of 16 timers
- Supports a maximum of six PWM channels
- Three timer modes—one-shot, restart, and PWM
- Maskable interrupt on timer expiration
- Programmable timer resolution as low as 41ms at 25MHz
- Maximum timeout period of 172sec at 25MHz
- Continuously updated reference counter

RISC timer table operations are based on a "tick" in the RISC internal timer that is programmed in the RISC controller configuration register (RCCR). The tick is a multiple of 1,024 general system clocks. The RISC timer tables have the lowest priority of all RISC microcontroller operations, so if it is so busy with other tasks that it is unable to service the timer during a tick interval, one or more of the timers might not be updated. This behavior can be used to estimate the worst-case loading of the microcontroller. The timer tables are configured in the RCCR, the timer table parameter RAM, the **SET TIMER** command that is issued to the CPCR, the timer event register, and the timer mask register.

16.2.7.4.1 RISC Timer Table Parameter RAM Memory Map. Two areas of internal RAM are used for the RISC timer tables—RISC timer table parameter RAM and the RISC timer table entries—which are shown in Figure 16-6. The RISC timer table parameter RAM area begins at the RISC timer base address and is used for the general timer parameters. See Table 16-4 for details.

Note: All references to registers in the parameter RAM table are actually implemented in the dual-port RAM area as a memory-based register.

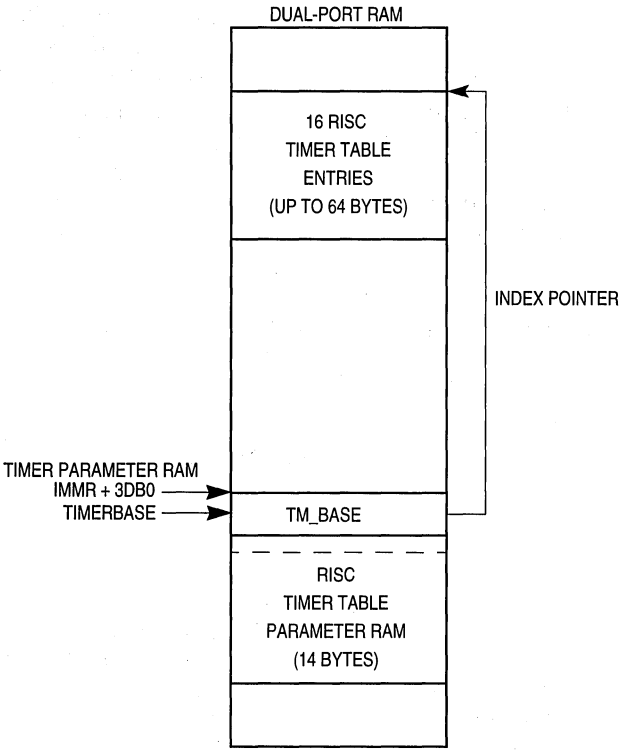


Figure 16-6. RISC Timer Table RAM Usage

Table 16-4. RISC Timer Table Parameter RAM Memory Map

ADDRESS	NAME	WIDTH	DESCRIPTION
Timer Base + 00	**TM_BASE**	Half-word	RISC Timer Table Base Address Index Pointer
Timer Base + 02	TM_PTR	Half-word	RISC Timer Table Pointer
Timer Base + 04	R_TMR	Half-word	RISC Timer Mode Register
Timer Base + 06	R_TMV	Half-word	RISC Timer Valid Register
Timer Base + 08	**TM_CMD**	Word	RISC Timer Command Register
Timer Base + 0C	**TM_CNT**	Word	RISC Timer Internal Counter

NOTE: You are only responsible for initializing the items in **bold**.
TimerBase = (IMMR & 0xFFFF0000) + 0x3DB0.

- TM_BASE—This index pointer contains a 16-bit offset from the beginning of the dual-port RAM to the location of your timer table entry. For the timer table entry area, you should allocate four bytes for each timer used. If you use all 16 timers, you must allocate 64 bytes in the timer table entry area. If you do not use all the timers, the timers should always be allocated in ascending order to save space. For example, if you only need two timers, then 8 bytes are required for the timer table entry area as long as you only enable RISC timers 0 and 1.

Note: The timer table entry area pointed to by the TM_BASE should always be aligned to a word boundary that is evenly divisible by four.

- TM_PTR—This index pointer is used exclusively by the RISC microcontroller to point to the next entry to be executed in the timer table. You should not modify this pointer.

- R_TMR—Only the RISC microcontroller uses this register to store the mode of the timer—one-shot (0) or restart (1). You should not modify this register. Instead, you should use the **SET TIMER** command in the CPCR to control the timer mode.

R_TMR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	TMR15	TMR14	TMR13	TMR12	TMR11	TMR10	TMR9	TMR8	TMR7	TMR6	TMR5	TMR4	TMR3	TMR2	TMR1	TMR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x3DB4															

TMR0–15—Timer 0–15

0 = No effect.
1 = Clears a bit in this register.

- R_TMV—Only the RISC microcontroller uses this register to determine whether or not a timer is currently enabled. If the corresponding timer is enabled, a bit is 1. You should not modify this register. You should use the **SET TIMER** command to enable a timer.

R_TMV

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	TMR15	TMR14	TMR13	TMR12	TMR11	TMR10	TMR9	TMR8	TMR7	TMR6	TMR5	TMR4	TMR3	TMR2	TMR1	TMR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x3DB6															

TMR0–15—Timer 0–15

0 = No effect.
1 = Clears a bit in this register.

- TM_CMD—This register is used as a parameter location when the **SET TIMER** command is issued. You should write this location prior to issuing the **SET TIMER** command. The bits of this register are defined as follows:

TM_CMD

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	V	R	PWM	RESERVED									TIMER NUMBER			
RESET	0	0	0	0									0			
R/W	R/W	R/W	R/W	R/W									R/W			
ADDR	(IMMR & 0xFFFF0000) + 0x3DB8															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	TIMER PERIOD															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x3DBA															

V—Valid

0 = Disables the timer.
1 = Enables the timer.

R—Restart

0 = One-shot timer operation.
1 = Automatic timer restart.

PWM—Pulse-Width Modulation Mode

0 = Normal mode.
1 = Pulse-width modulation.

Bits 3–11—Reserved

These bits are reserved and should be set to 0.

TIMER NUMBER

This bit is the value from zero to 15 that signifies timer configuration.

TIMER PERIOD

This bit is the 16-bit timeout value of the timer. The maximum value is 65,536 and is encoded as 0x0000.

- TM_CNT—This register is a tick counter that the microcontroller updates after each tick or after the timer table is scanned. It is updated if the microcontroller's internal timer is enabled, regardless of whether any of the 16 timers are enabled, and it can be used to track the number of ticks the microcontroller receives and responds to.

16.2.7.4.2 RISC Timer Table Entries. The 16 timers are located in the block of memory that TM_BASE points to and each timer occupies 4 bytes. The first half-word forms the initial value of the timer written when the **SET TIMER** command is executed. The next half-word is the current value of the timer that gets decremented until it reaches zero. You should not modify these locations because they should only be used for debugging purposes.

16.2.7.4.3 The SET TIMER Command. This command is issued to the CPCR and is used to enable, disable, and configure the 16 timers in the RISC timer table. The 0x0851 value should be written to the CPCR, but before doing that you should set up the TM_CMD value.

16.2.7.4.4 PWM Mode. Each pair of timers can be used to generate a pulse-width modulation waveform on one of the port B pins. A maximum of six channels are supported. The first timer (even numbered) is used to control the duty-cycle time of the waveform. In the register above, the TIMER PERIOD entry should be set to the high period of the waveform and the PWM and V bits should be set to 1. The second timer (odd numbered) is used to control the cycle time. The TIMER PERIOD entry should be set to the preferred cycle time, the PWM bit should be set to zero, and the R and V bits should be set to 1. Table 16-5 shows the port B pin assignments for the PWM mode. The respective port B pins should be configured as general-purpose outputs in the PBDIR in **Section 16.14.6.3 Port B Data Direction Register.**

Table 16-5. PWM Channel Pin Assignments

TIMER PAIRS	PORT B PIN
0 and 1	Port B[23]
2 and 3	Port B[22]
4 and 5	Not Available
6 and 7	Not Available
8 and 9	Port B[19]
10 and 11	Port B[18]
12 and 13	Port B[17]
14 and 15	Port B[16]

16.2.7.5 RISC TIMER EVENT REGISTER. The 16-bit RISC timer event register (RTER) is used to report events recognized by the 16 timers and to generate interrupts. However, an interrupt is only generated if the RISC timer table bit is set in the CPM interrupt mask register. More than one bit can be cleared at a time.

RTER

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	TMR15	TMR14	TMR13	TMR12	TMR11	TMR10	TMR9	TMR8	TMR7	TMR6	TMR5	TMR4	TMR3	TMR2	TMR1	TMR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x9D6															

TMR0–15—Timer 0–15

0 = No effect.
1 = Clears a bit in the register.

16.2.7.6 RISC TIMER MASK REGISTER. This 16-bit read/write RISC timer mask register (RTMR) is used to enable interrupts that can be generated in the RTER. If a bit is set, it enables the corresponding interrupt in the RTER. If a bit is cleared, this register masks the corresponding interrupt in the RTER. However, an interrupt is only generated if the R-TT bit is set in the CPM interrupt mask register (CIMR), which is described in **Section 16.15.5.3 CPM Interrupt Mask Register**.

RTMR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	TMR15	TMR14	TMR13	TMR12	TMR11	TMR10	TMR9	TMR8	TMR7	TMR6	TMR5	TMR4	TMR3	TMR2	TMR1	TMR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x90A															

TMR0–15—Timer 0–15

0 = Masks the corresponding interrupt in the RTER.
1 = Enables the corresponding interrupt in the RTER.

16.2.7.7 RISC TIMER INITIALIZATION SEQUENCE EXAMPLE. Follow these steps to initialize the RISC timers:

1. Configure the RCCR to determine the preferred tick interval that will be used for the entire timer table. Normally, you can set the TIME bit at this time. However, it can be set later if all RISC timers must be synchronized.
2. Determine the maximum number of timers to be located in the timer table. Configure the TM_BASE pointer of the RISC timer table parameter RAM to point to a location in the dual-port RAM with 4 × N bytes available, where N is the number of timers. If N is less than 16, use timer 0 through timer N–1 to save space.
3. Clear the TM_CNT counter of the RISC timer table parameter RAM to show how many ticks have elapsed since the RISC internal timer was enabled (optional).
4. Clear the RTER, if it is not already cleared. A one clears this register.
5. Configure the RTMR to enable the timers that need to generate interrupts.
6. Set the R-TT bit in the CIMR to generate interrupts to the system. Make sure the CPM interrupt controller is properly initialized.
7. Configure the TM_CMD register of the RISC timer table parameter RAM. At this point, determine whether a timer is to be enabled or disabled, one-shot or restart, and what its timeout period should be. If the timer is being disabled, all parameters besides the timer number are ignored.
8. Issue the **SET TIMER** command by writing 0x0851 to the CPCR.
9. Repeat steps 7 and 8 for each timer to be enabled or disabled.

As an example, the following sequence demonstrates how the RISC timer 0 is initialized to generate an interrupt approximately every second using a 25MHz general system clock:

1. Write 111111 to the TIMEP field of the RCCR to generate the slowest clock. This value generates a tick every 65,536 clocks, which is every 2.6 milliseconds at 25MHz.
2. Configure the TM_BASE pointer of the RISC timer table parameter RAM to point to a location in the dual-port RAM with 4 bytes available. Assuming that the beginning of dual-port RAM is available, write 0x0000 to TM_BASE.
3. Write 0x0000 to the TM_CNT counter of the RISC timer table parameter RAM to see how many ticks have elapsed since the RISC internal timer was enabled (optional).
4. Write 0xFFFF to the RTER to clear any previous events.
5. Write 0x0001 to the RTMR to enable RISC timer 0 to generate an interrupt.
6. Write 0x00020000 to the CPM interrupt mask register so the RISC timers will generate a system interrupt. Initialize the CPM interrupt configuration register.
7. Write 0xC0000EE6 to the TM_CMD register of the RISC timer table parameter RAM. This enables RISC timer 0 to timeout after 3,814 (decimal) ticks. The timer automatically restarts after it times out.
8. Write 0x0851 to the CPCR to issue the **SET TIMER** command.
9. Set the TIME bit in the RCCR to operate the RISC timer.

16.2.7.8 RISC TIMER INTERRUPT HANDLING EXAMPLE. The following sequence shows what is normally written in an interrupt handler for the RISC timer tables:

1. Once an interrupt occurs, read the RTER to see which timers have caused interrupts. The RISC timer event bits are usually cleared by this time.
2. Issue any additional **SET TIMER** commands now or later, whichever you prefer. You do not have to do anything if the timer is automatically being restarted for a repetitive interrupt.
3. Clear the R-TT bit in the CPM interrupt status register.
4. Execute the **rfi** instruction.

16.2.7.9 RISC TIMER TABLE ALGORITHM. The RISC microcontroller scans the timer table once every tick. For each valid timer in the table, the microcontroller decrements the count and checks for a timeout and if no timeout occurs, it moves to the next timer. If a timeout does occur, the microcontroller sets the corresponding event bit in the RISC timer event register. Then it checks to see if the timer needs to be restarted and if it does, it leaves the R_TMV register and resets the current count to the initial count. Otherwise, it clears the R_TMV register. Once the timer table is scanned, the microcontroller updates the TM_CNT value in the RISC timer table parameter RAM and stops working on the timer tables until the next tick. If a **SET TIMER** command is issued, the microcontroller makes the appropriate modifications to the timer table and parameter RAM, but does not scan the timer table until the next tick of the internal timer. If you modify the RISC timer table, execute the **SET TIMER** command to synchronize the timers so that the microcontroller will operate properly.

16.2.7.10 USING THE TIMERS TO TRACK MICROCONTROLLER LOADING. The following sequence of steps is a method for using the 16 timers to determine if the microcontroller ever exceeds the 96% utilization level during a tick interval. Removing the timers adds a 4% margin to the microcontroller's utilization level, but an aggressive user can use this technique to push the microcontroller performance to its limit. You should use the standard initialization sequence, but incorporate the following steps:

1. Program the tick of the RISC microcontroller timers to be 1,024 × 16 = 16,384.
2. Disable microcontroller timer interrupts, as required.
3. Using the **SET TIMER** command, initialize all 16 RISC microcontroller timers to have a timer period of 0x0000, which equals 65,536.
4. Program one of the four general-purpose timers to increment once every tick. The general-purpose timer should be free-running and have a timeout of 65,536.
5. After a few hours of operation, compare the general-purpose timer to the current count of RISC microcontroller timer 15 and if the difference between them exceeds two ticks, the microcontroller has, during some tick interval, exceeded the 96% utilization level.

Note: The general-purpose timers are up-counters, but the RISC microcontroller timers are down-counters. You should take this under consideration when comparing timer counts.

16.3 DIGITAL SIGNAL PROCESSING

Many embedded control applications require DSP-style algorithm implementations, such as finite impulse response (FIR) filters with or without adaptive equalization, data compression, and scrambling. These are written in software on the MPC823 and do not require your system to have a separate DSP processor, which would cost you more and consume more power. The communication processor module provides the additional power you need for those applications.

The RISC microcontroller's instruction set (described in **Appendix B MPC823 Instruction Set**) supports high-performance multiply and accumulate (MAC) operation as well as special addressing modes that are essential to efficient DSP algorithm implementation. The RISC microcontroller runs concurrently with the core and increases the core's bandwidth left for other system tasks. The system can take advantage of this increased core bandwidth by lowering the system clock frequency and voltage, which decreases the amount of power that is consumed.

16.3.1 Features

The following list summarizes the features of MPC823 DSP:

- 16 × 16-bit multiply and accumulate
- Load/store with automatic post increment/decrement
- DSP routine library provides 11 basic building blocks for implementation of V.34bis and 56K

16.3.2 DSP Operation

There are three layers to DSP functionality—hardware, firmware, and software. You only need to construct the software layer to generate an application.

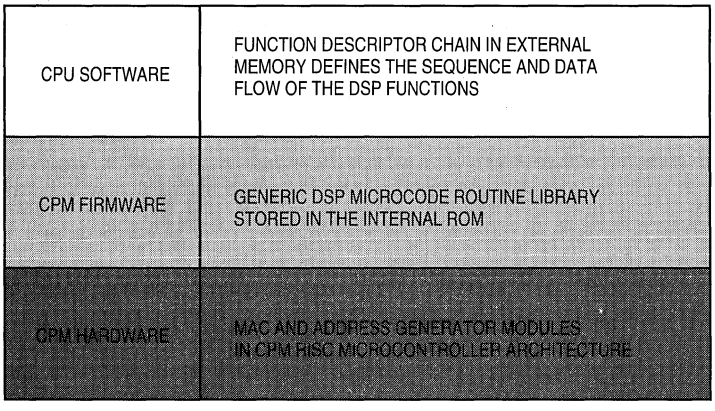


Figure 16-7. DSP Functionality Implementation

16.3.2.1 HARDWARE. The RISC microcontroller's hardware contains special DSP processing units, such as a multiplier and accumulator that is capable of handling real or complex numbers, and an address generator that can access cyclic buffer structures in dual-port RAM.

16.3.2.2 SOFTWARE. Your software interfaces to the DSP via the function descriptor that is described in system memory. The function descriptor defines the sequence and data flow of your DSP task.

16.3.2.3 FIRMWARE. The RISC microcontroller's firmware is a set of DSP functions that have been compiled to form a library of basic building blocks and each function within the library is implemented by a microcode routine stored in the internal ROM. In addition, a software interface is defined that enables parameters to be passed between the core and communication processor module. Several functions can be chained together to reduce software intervention and interrupt rates, assuming that all data structures reside in the dual-port RAM. Table 16-6 lists the DSP functions that are included in the library.

Table 16-6. DSP Functions

FUNCTION	OPCODE	INPUT	COEFFICIENT	OUTPUT	APPLICATION
FIR1	00001	Real	Real	Real	Decimation, RX Interpolation
FIR2	00010	Complex	Real	Complex	TX Filter, RX Filter
FIR3	00011	Complex	Complex	Real/Complex	EC Computation, Equalizer
FIR5	00011	Complex	Complex	Real/Complex	Fractionally Spaced Equalizer
FIR6	00110	Real	Complex	Complex	—
IIR	00111	Real	Real	Real	Biquad Filter
MOD	01000	Complex	Complex	Real/Complex	TX Modulation
DEMOD	01001	Real	Complex	Complex	RX Demodulation
LMS1	01010	—	—	—	EC Update, Equalizer Update (T/2, T/3)
LMS2	01011	—	—	—	Equalizer Update (2T/3)
WADD	01100	Real	—	Real	Interpolation

16.3.3 Programming the DSP Functions

Similar to the SCC2 buffer descriptor, a function descriptor (FD), is used to specify the DSP function and pass the parameters. A table of such descriptors forms a circular queue with a programmable length. The descriptors are stored in external memory. There are two function descriptor tables (also referred to as chains)—one for the transmitter and one for the receiver. The core prepares a chain of function descriptors in the system memory and a special host command notifies the RISC microcontroller when to execute the chain. A maskable interrupt is generated once the chain is completed. As shown in Figure 16-8, the pointer to the transmit (TX) chain must be written into the FDBASE field of the DSP2 parameter RAM and the pointer to the receive (RX) chain must be written into DSP1.

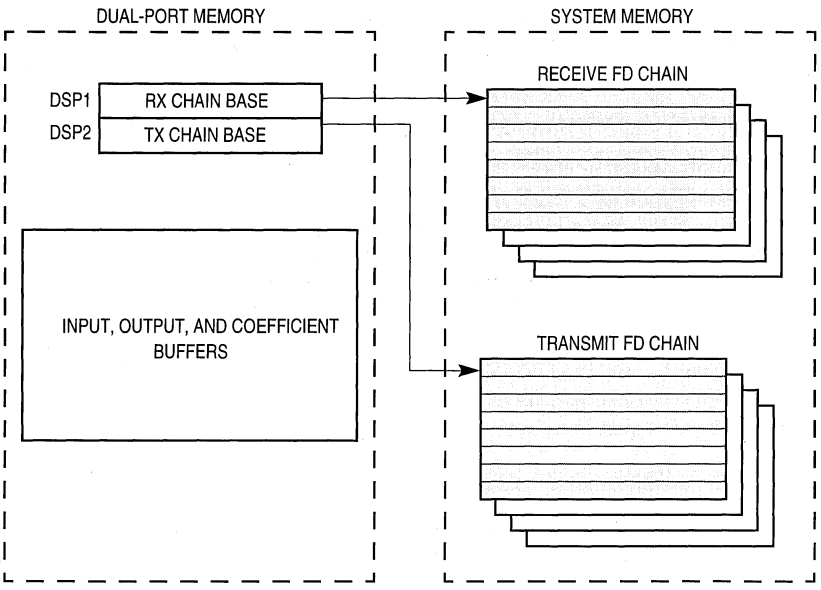


Figure 16-8. DSP Function Descriptor Operation

16.3.3.1 DATA REPRESENTATION. The inputs, coefficients, and outputs are represented by 16-bit, fixed-point, 2's complement numbers. A real number is represented by a single 16-bit half-word, as shown in Figure 16-8. Its value is between -1 (0x8000) to +1 (0x7FFF) and you must scale your data to fit this range. A complex number is represented by a pair of 16-bit half-words—one word for the imaginary component and one for the real component as shown in Figure 16-8. They must be scaled to fit in the -1 and +1 range.

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	S	REAL FRACTION														

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

16.3.3.2 MODULO ADDRESSING. The input and output buffers are circular within a certain programmable size that must be a multiple of 2^k. The base address of the circular buffer must be aligned on its natural size boundary. For example, if your input buffer size is 128 bytes, your base address must be aligned on a 128-byte boundary. In other words, the lower boundary (base address) of a circular buffer containing modulus (M) bytes must have zeros in the k LSBs of the base address, where $2^k \geq M$, and therefore must be a multiple of 2^k. The upper boundary is the lower boundary, plus the size minus one (base address + M-1). Once M is chosen, a sequential series of memory blocks (each of length 2^k) is created where these circular buffer can be located. If $M < 2^k$, there is a 2^k-M space between the sequential M-sized circular buffers and M should be a multiple of four. See Figure 16-9 for details.

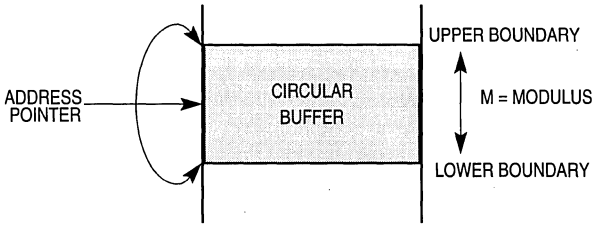


Figure 16-9. Circular Buffer

16.3.3.2.1 DSP Function Descriptors. Each function descriptor is composed of eight 16-bit half-words. The first half-word contains the function opcode as well as status and control bits and the other half-words contain the function's parameter packet. Each function has its own parameter packet.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	RES	W	I	RESERVED							OPCODE				
OFFSET + 2	PARAMETER 1															
• • •	• • •															
• • •	• • •															
• • •	• • •															
OFFSET + E	PARAMETER 7															

S—STOP

0 = Do not stop after executing this function descriptor.
1 = Stop after executing this function descriptor.

Bits 1 and 4–10—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Function Descriptor in Table)

0 = This is not the last function descriptor in the function descriptor table.
1 = This is the last function descriptor in the function descriptor table. After this buffer has been used, the communication processor module processes the first function descriptor that the FDBASE index pointer points to in the table. The number of function descriptors in this table is programmable and determined only by the W bit and the overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

OPCODE—Function Operation Code

This field specifies the function that should be executed. Some of these bits are reserved so they can be expanded by tapping into RAM and executing a routine. See Table 16-6 for more information.

16.3.3.2.2 DSP Parameter RAM Memory Map. Each section in the dual-port RAM is associated with each DSP chain and can be used for parameter storage or as a scratchpad. The FDBASE index pointer defines the place in system memory for the function descriptor chain to start. It should be 16-byte aligned. Also, the FDBASE index pointer should be initialized before the **INIT_DSP** command is issued. See Table 16-7 for DSP parameter RAM memory map details.

Table 16-7. DSP Parameter RAM Memory Map

ADDRESS	NAME	WIDTH	DESCRIPTION
DSP Base + 0x00	**FDBASE**	Word	Function Descriptor Table Base Address
DSP Base + 0x04	FD_PTR	Word	Function Descriptor Pointer
DSP Base + 0x08	DSTATE	Word	DSP State
DSP Base + 0x10	DSTATUS	Half-word	Current Function Descriptor Status
DSP Base + 0x12	I	Half-word	Current Function Descriptor Number of Iterations
DSP Base + 0x14	TAP	Half-word	Current Function Descriptor Number of TAPs
DSP Base + 0x16	CBASE	Half-word	Current Function Descriptor Cbase Pointer
DSP Base + 0x18	—	Half-word	Current Function Descriptor Sample Buffer Size-1
DSP Base + 0x1A	XPTR	Half-word	Current Function Descriptor Pointer to Sample Pointer
DSP Base + 0x1C	—	Half-word	Current Function Descriptor Output Buffer Size-1
DSP Base + 0x1E	YPTR	Half-word	Current Function Descriptor Pointer to Output Buffer Pointer
DSP Base + 0x20	M	Half-word	Current Function Descriptor Sample Buffer Size-1
DSP Base + 0x22	—	Half-word	Current Function Descriptor Sample Buffer Pointer
DSP Base + 0x24	N	Half-word	Current Function Descriptor Output Buffer Size-1
DSP Base + 0x26	—	Half-word	Current Function Descriptor Output Buffer Pointer
DSP Base + 0x28	K	Half-word	Current Function Descriptor Coefficient Buffer Size-1
DSP Base + 2A	—	Half-word	Current Function Descriptor Coefficient Buffer Pointer

NOTE: You are only responsible for initializing the items in bold.
DSP1 Base = (IMMR & 0xFFFF0000) + 0x3EC0 and DSP2 base = (IMMR & 0xFFFF0000) + 0x3FC0.

FDBASE—Function Descriptor Table Base Address

This index pointer defines the location in system memory where the function descriptor starts. However, you must initialize it.

FD_PTR—Function Descriptor Pointer

This pointer points to the current function descriptor address. It is only used by the RISC microcontroller, so you do not need to modify it in any way.

DSTATE—Current State

This bit defines the internal state of the RISC microcontroller. It is only used by the RISC microcontroller, so you do not need to modify it in any way.

DSTATUS—Current Function Descriptor Status

This bit defines the current status of the current function descriptor. It is only used by the RISC microcontroller, so you do not need to modify it in any way.

I—Current Function Descriptor Number of Iterations

This bit is only used by the RISC microcontroller, so you do not need to modify it in any way.

TAP—Current Function Descriptor Number of Taps
The bit is only used by the RISC microcontroller, so you do not need to modify it in any way.

CBASE—Current Function Descriptor Cbase
The bit defines the current function descriptor base address of the coefficients. It is only used by the RISC microcontroller, so you do not need to modify it in any way.

XPTR—Current Function Descriptor Pointer to Sample Pointer
The bit is only used by the RISC microcontroller, so you do not need to modify it in any way.

YPTR—Current Function Descriptor Pointer to Output Buffer Pointer
The bit is only used by the RISC microcontroller, so you do not need to modify it in any way.

M—Current Function Descriptor Sample Buffer Size-1
The bit is only used by the RISC microcontroller, so you do not need to modify it in any way.

N—Current Function Descriptor Output Buffer Size-1
The bit is only used by the RISC microcontroller, so you do not need to modify it in any way.

K—Current Function Descriptor Coefficient Buffer Size-1
The bit is only used by the RISC microcontroller, so you do not need to modify it in any way.

16.3.3.2.3 DSP Commands. The following commands are issued to the CPM command register (CPCR) and they are defined in detail in Table 16-6.

- **INIT DSP CHAIN**—Deactivates the corresponding chain. The function descriptor pointer is initialized to the starting address provided in the function descriptor table.
- **START DSP CHAIN**—Activates the corresponding chain.

16.3.3.3 DSP EVENT REGISTER. For DSP interrupts, the memory-mapped SDMA status register (SDSR) is used to generate maskable interrupts to the core. An interrupt is set when the function finishes executing if the I bit is set in the function descriptor. There are two interrupt events—DSP1 and DSP2—that are each associated with a corresponding chain. A bit is reset by writing a 1 (writing a zero has no effect) and more than one bit can be reset at a time.

SDSR

BIT	0	1	2	3	4	5	6	7
FIELD	SBER	RESERVED					DSP2	DSP1
R/W	R/W	R/W					R/W	R/W
RESET	0	0					0	0
ADDR	(IMMR & 0xFFFF0000) + 0x908							

SBER—SDMA Channel Bus Error (SDMA function)

When set, this bit indicates that an error caused the SDMA channel to terminated during a read or write cycle. The SDMA bus error address can be read from the SDMA address register, as shown in **Section 16.5.2.4 SDMA Address Register**.

Bits 1–5—Reserved

These bits are reserved and should be set to 0.

DSP2—DSP Chain 2 Transmitter Interrupt (DSP function)

This bit is set when the chain 2 function finishes executing. However, the I bit must be set in the function descriptor.

DSP1—DSP Chain 1 Receiver Interrupt (DSP function)

This bit is set when the chain 1 function finishes executing. However, the I bit must be set in the function descriptor.

16.3.3.4 DSP MASK REGISTER. The 8-bit read/write SDMA mask register (SDMR) is used to mask the DSP interrupts and has the same bit format as the SDSR. If a bit in the SDMR is a 1, the corresponding interrupt in the SDSR is enabled and if it is zero, the corresponding interrupt is masked. This register is cleared by reset.

SDMR

BIT	0	1	2	3	4	5	6	7
FIELD	SBER	RESERVED					DSP2	DSP1
R/W	R/W	R/W					R/W	R/W
RESET	0	0					0	0
ADDR	(IMMR & 0xFFFF0000) + 0x90C							

SBER—SDMA Channel Bus Error (SDMA function)

0 = Disable the interrupt.
1 = Enable the interrupt.

Bits 1–5—Reserved

These bits are reserved and should be set to 0.

DSP1—DSP Chain 1 Receiver Interrupt (DSP function)

0 = Disable the DSP chain 1 interrupt.
1 = Enable the DSP chain 1 interrupt.

DSP2—DSP Chain 2 Transmitter Interrupt (DSP function)

0 = Disable the DSP chain 2 interrupt.
1 = Enable the DSP chain 2 interrupt.

16.3.3.5 DSP IMPLEMENTATION. There are basically two ways to implement a DSP task—run C code on the core or use the communication processor module functions. Figure 16-10 illustrates an example section of a V.32 modem's transmit (TX) data pump flow. The TX filter is composed of three FIR2 subfilters.

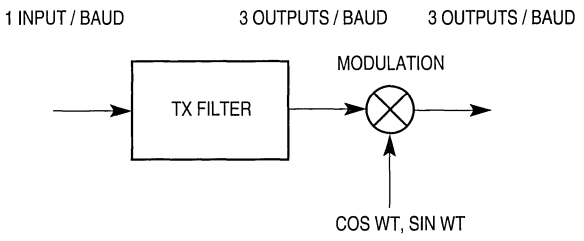


Figure 16-10. DSP Implementation Example

To implement this DSP task with C code on the core, it takes 476 core instructions (371 for the filter and 105 for the modulation) to execute the code. Repeating that 2,400 times a second consumes 1.14MIPS (476 x 2,400) of the core. To implement a task using the CPM functions, the software builds a static function descriptor structure composed of two chained functions—a FIR2 and a MOD. The core activates the RISC microcontroller to execute those functions by sending a single write to the CPM command register. Using an interrupt, the communication processor module then signals that the process has completed. The communication processor module executes the functions twice as efficiently as the core, which results in 0.55 CPM MIPS and very few core cycles.

The TX filter is implemented by executing three subfilters each time a new sample is received. This is accomplished by invoking FIR2 with a three-iteration count and autoincrement of the input sample pointer when the function is completed. FIR2 writes the three results into the output buffer, which is also the modulation input buffer. Modulation is accomplished by invoking MOD with a three-iteration count. The input pointer is autoincremented with each iteration.

16.3.3.5.1 DSP Programming Example (Core Only).

```
void tx_filter ()
{
   S16 *coefr
   S16 *samplr, *sampli
   S16 *coefend;
   S32 filtoutr, filtouti;
   U8 subcount, sampleindex;
   extern S16 mult(S16 p1, S16 p2);      /* in-line invocation */

   coefr=txfiltcoef_str;
   coefend=txfiltcoef_end;
   samplr=&txfiltdly[REAL][txfiltptr];
   sampli=&txfiltdly[IMAG][txfiltptr];
   sampleindex=0;
   while (coefr<coefend) {
      filtoutr=filtouti=0;
      subcount=0;
      while (subcount<TXSUBFILTLEN) {
         filtoutr+=mult(*coefr, *samplr--);
         filtouti+=mult(*coefr++, *sampli--);
      }
      samplr=&txfiltdly[REAL][txfiltptr];
      sampli=&txfiltdly[IMAG][txfiltptr];
      modbuff[REAL][sampleindex]= filtoutr ;
      modbuff[IMAG][sampleindex++]= filtouti;
   }
}

void modulator ()
{

   U8 i;
   S32 termrnd;
   extern S16 mult(S16 p1, S16 p2);      /* in-line invocation */

   i=0;
   while (i<SAMPLE_PER_T) {
      sigout[i]= mult(sn1800[REAL][cosindx], modbuf[REAL][i]) -
               mult(sn1800[IMAG][cosindx], modbuf[IMAG][i]);
      cosindx++;
      if (cosindx==SIN1800TBL_LEN)cosindx=0;
      i++;
   }

void main ()
{
        *
        *
   tx_filter();
   modulator();
        *
        *
}
```

16.3.3.5.2 DSP Programming Example (Core and CPM). Figure 16-11 illustrates how the data buffer and function descriptor data is organized in system memory and dual-port RAM. The function descriptor resides in system memory and all input, ouput, coefficient data must reside in dual-port RAM. The transmit and modulation FD chain can reside in either system memory or dual-port RAM.

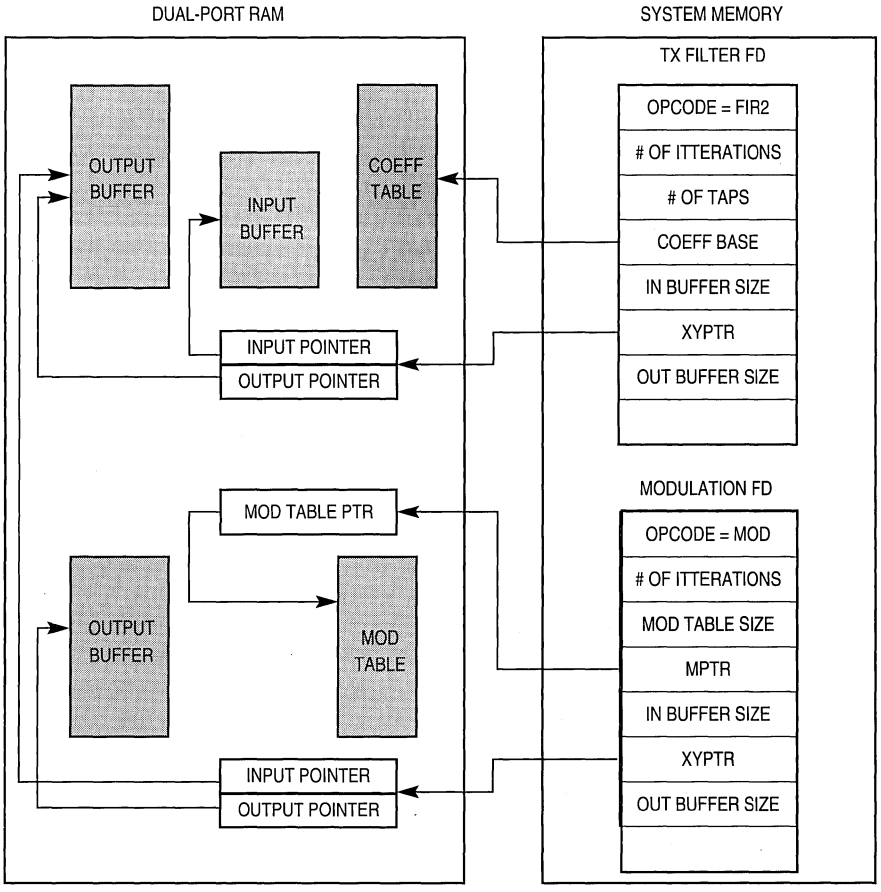


Figure 16-11. Core and CPM Implementation

```
/* Buffer Descriptors */
typedef struct dsp_fd {
        unsigned short  status;
        unsigned short  parameter[7];
} DSP_FD;

#define WRAP    0x2000          /* wrap bit */
#define INTR    0x1000          /* interrupt on completion */


/* define for function opcodes */
#define FIR_2   0x0102  /* FIR2  filter */
#define MOD     0x0008  /* Modulation function opcode */

/* Initialize a static fd table for 2 functions */
DSP_FD  filters[2]= {
                 { FIR_2,P11,P12, , P17}
                ,{(WRAP | INTR | MOD),P21,P22, , P27}
};

void main()
{
        /* Setup FD chain pointer */
        DSP2_Base. Fdbase = filters;
        *
        /* issue command to CPM to start processing the fd chain */
        issue_command( START_FD );
        *
        *
        *
}
```

16.3.4 DSP On-Chip Library Functions

The DSP library is an easy way to implement DSP functions. It consists of the following functions:

- FIR1—Finite impulse response 1
- FIR2—Finite impulse response 2
- FIR3—Finite impulse response 3
- FIR5—Finite impulse response 5
- FIR6—Finite impulse response 6
- IIR—Infinite impulse response
- MOD—Modulation
- DEMOD—Demodulation
- LMS1—Least mean squared 1
- LMS2—Least mean squared 2
- WADD—Weighted vector addition

16.3.4.1 FIR1–REAL C, REAL X, AND REAL Y. The FIR1 function implements a basic FIR filter with k real coefficients, real input samples, and real output. The input data is in a circular buffer with size M+1 and the output data is in a circular buffer with size N+1.

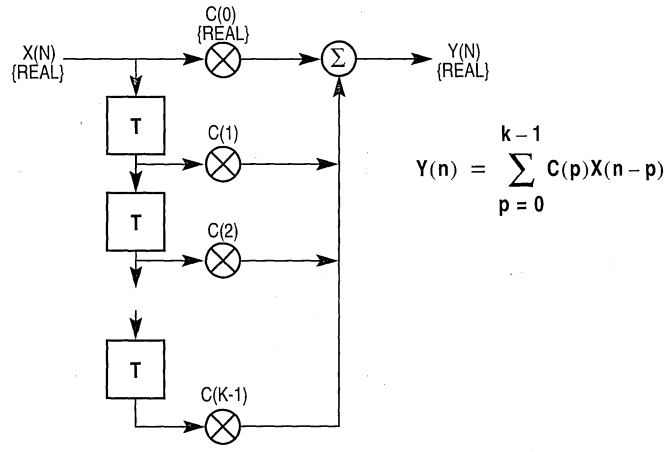


Figure 16-12. FIR1 Implementation Example

16.3.4.1.1 Coefficients and Sample Data Buffers. The coefficients vector occupies k 16-bit half-words in memory and C(0) is stored in the first location. The sample input buffer is a cyclic buffer containing M+1 bytes. Each sample is a 16-bit word and the new sample is stored in the address that follows the previous sample. The output buffer is a cyclic buffer that contains N+1 bytes. Each output is a 16-bit half-word and the new output is stored in the address that follows the previous output.

COEFFICIENTS
C(0)
C(1)
C(2)
C(k-1)

2K bytes

INPUT SAMPLES
*
*
x(N-k+1)
*
*
x(N-2)
x(N-1)
x(N)

M + 1 bytes

OUTPUT
*
*
*
*
Y(N-k+1)
*
*
Y(N-2)
Y(N-1)
Y(N)

N + 1 bytes

Figure 16-13. FIR1 Coefficients and Sample Data Buffers

16.3.4.1.2 FIR1 Function Descriptor. The FIR1 function descriptor bit table is described below.

	0	1	2	3	4	5	6 7	8	9 10	11 12 13 14 15
OFFSET + 0	S	RES	W	I	RES	IALL	INDEX	PC	RES	OPCODE
OFFSET + 2	I									
OFFSET + 4	K									
OFFSET + 6	CBASE									
OFFSET + 8	M									
OFFSET + A	XYPTR									
OFFSET + C	N									
OFFSET + E	RESERVED									

The first half-word is composed of the following bits:

S—STOP

0 = Do not stop after executing this function descriptor.
1 = Stop after executing this function descriptor.

Bits 1, 4, and 9–10—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Function Descriptor in Table)

0 = This is not the last function descriptor in the function descriptor table.
1 = This is the last function descriptor in the function descriptor table. After this buffer has been used, the CPM processes the first function descriptor that the FDBASE index pointer points to in the table. The number of function descriptors in this table are programmable and determined only by the W bit and overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

IALL— Auto-Increment X For All Iterations

0 = The X (input) data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field after the last iteration.
1 = The X data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field after each iteration.

INDEX— Auto-Increment Index

00 = The X (input) pointer is not incremented.
01 = The X (input) pointer is incremented by one sample.
10 = The X (input) pointer is incremented by two samples.
11 = The X (input) pointer is incremented by three samples.

PC— Preset Coefficients Pointer

0 = The coefficients pointer is not preset after each iteration.
1 = The coefficients pointer is preset after each iteration to the CBASE pointer.

OPCODE—Function Operation Code

This field specifies the function to be executed. See Table 16-6 to get the value for this field.

16.3.4.1.3 FIR1 Parameter Packet. The FIR1 parameter packet is composed of seven 16-bit half-words and described in the following table.

Table 16-8. FIR1 Parameter Packet

ADDRESS	NAME	DESCRIPTION
Half-word 1	I	Number of Iterations
Half-word 2	K	Number of TAPs-1. The number of taps should be a multiple of four
Half-word 3	CBASE	Filter Coefficients Vector Base Address Pointer
Half-word 4	M	Samples Buffer Size-1. The minimum sample buffer size is 8 (4 samples)
Half-word 5	XYPTR	Pointer to a structure composed of the input sample data pointer and the output buffer pointer
Half-word 6	N	Output Buffer Size-1. The minimum output buffer size is 4 (2 outputs)
Half-word 7	RES	Reserved

16.3.4.1.4 Application Example. The FIR1 is used in decimation and RX interpolation. For example, the following function descriptor structure can be used to implement a 2 to 1 decimation.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	0	W	I	0	1	10		1	0	0	00001				
OFFSET + 2	I=3 (THREE ITERATIONS)															

16.3.4.2 FIR2–REAL C, COMPLEX X, AND COMPLEX Y. The FIR2 function implements a basic FIR filter with k real coefficients, complex input samples, and complex output. The input data is in a circular buffer with size M+1 and the output data is in a circular buffer with size N+1.

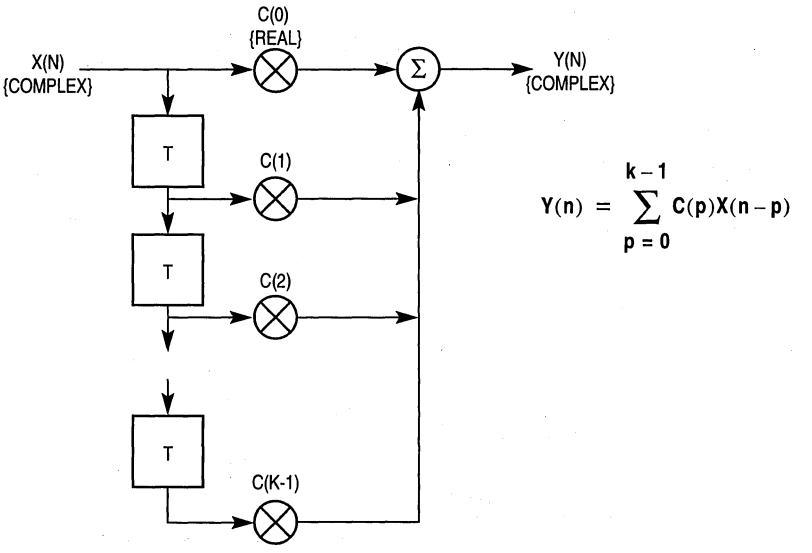


Figure 16-14. FIR2 Implementation Example

16.3.4.2.1 Coefficients and Sample Data Buffers. The coefficients vector occupies k 16-bit half-words in memory and C(0) is stored in the first location. The sample input buffer is a cyclic buffer that contains M+1 bytes and each input sample is two 16-bit half-words (real and imaginary components). The new sample is stored in the address that follows the previous sample. The output buffer is a cyclic buffer containing N+1 bytes. Each output is two 16-bit half-words (real and imaginary components). The new output is stored in the address that follows the previous output.

COEFFICIENTS
C(0)
C(1)
C(2)
C(k-1)

2K bytes

INPUT SAMPLES
*
*
imag {x(n-k+1)}
real {x(n-k+1)}
*
imag {x(n-2)}
real{x(n-2)}
imag{x(n-1)}
real{x(n-1)}
imag{x(n)}
real{x(n)}

M + 1 bytes

OUTPUT
*
*
*
imag{Y(n-k+1)}
real{Y(n-k+1)}
*
*
imag{Y(n-2)}
real{Y(n-2)}
imag{Y(n-1)}
real{Y(n-1)}
imag{Y(n)}
real{Y(n)}

N + 1 bytes

Figure 16-15. FIR2 Coefficients and Sample Data Buffers

16.3.4.2.2 FIR2 Function Descriptor. The FIR2 function descriptor bit table is described below.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	RES	W	I	RES	IALL	INDEX		PC	RES		OPCODE				
OFFSET + 2	I															
OFFSET + 4	K															
OFFSET + 6	CBASE															
OFFSET + 8	M															
OFFSET + A	XYPTR															
OFFSET + C	N															
OFFSET + E	RESERVED															

The first half-word is composed of the following bits:

S—STOP

0 = Do not stop after executing this function descriptor.
1 = Stop after executing this function descriptor.

Bits 1, 4, 9, and 10—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Function Descriptor in Table)

0 = This is not the last function descriptor in the function descriptor table.
1 = This is the last function descriptor in the function descriptor table. After this buffer has been used, the CPM processes the first function descriptor that the FDBASE index pointer points to in the table. The number of function descriptors in this table are programmable and determined only by the W bit and overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

IALL— Auto-Increment X For All Iterations

0 = The X (input) data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field after the last iteration.
1 = The X data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field after each iteration.

INDEX— Auto-Increment Index

00 = The X (input) pointer is not incremented.
01 = The X (input) pointer is incremented by one sample.
10 = The X (input) pointer is incremented by two samples.
11 = The X (input) pointer is incremented by three samples.

PC— Preset Coefficients Pointer

0 = The coefficients pointer is not preset after each iteration.
1 = The coefficients pointer is preset after each iteration to CBASE pointer.

OPCODE—Function Operation Code

This field specifies the function to be executed. See Table 16-6 to get the value for this field.

16.3.4.2.3 FIR2 Parameter Packet. The FIR2 parameter packet is composed of seven 16-bit half-words and described in the table below.

Table 16-9. FIR2 Parameter Packet

ADDRESS	NAME	DESCRIPTION
Half-word 1	I	Number of Iterations.
Half-word 2	K	Number of TAPs-1.
Half-word 3	CBASE	Filter Coefficients Vector Base Address Pointer
Half-word 4	M	Samples Buffer Size-1. The minimum sample buffer size is 8 (4 samples)
Half-word 5	XYPTR	Pointer to a structure composed of the input sample data pointer and the output buffer pointer
Half-word 6	N	Output Buffer Size-1. The minimum output buffer size is 8 (2 outputs)
Half-word 7	RES	Reserved

16.3.4.2.4 Application Example. The FIR2 function is used in the TX and RX filters. For example, the following function descriptor structure can be used to implement the TX filter.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	0	W	I	0	0	01		0	0	0	00010				
OFFSET + 2	I=3 (THREE ITERATIONS)															

16.3.4.3 FIR3–COMPLEX C, COMPLEX X, AND REAL/COMPLEX Y. The FIR3 function implements a basic FIR filter with k complex coefficients, complex input samples, and real or complex output. The input data is in a circular buffer with size M+1 and the output data is in a circular buffer with size N+1.

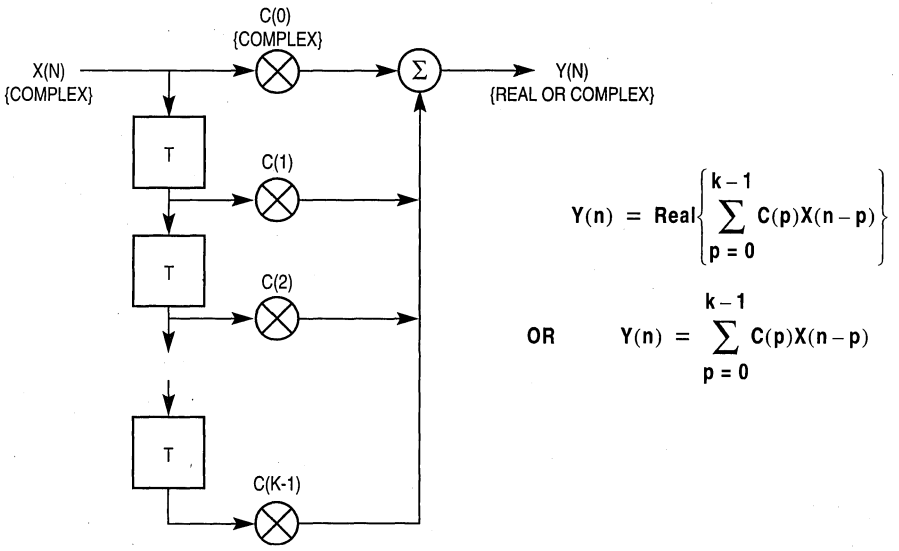


Figure 16-16. FIR2 Implementation Example

16.3.4.3.1 Coefficients and Sample Data Buffers. The coefficients vector occupies k pairs of 16-bit half-words (real and imaginary components) in memory and C(0) is stored in the first location. The sample input buffer is a cyclic buffer containing M+1 bytes and each input sample is two 16-bit half-words (real and imaginary components). The new sample is stored in the address that follows the previous sample. The output buffer is a cyclic buffer that contains N+1 bytes and each output is two 16-bit half-words (real and imaginary components). The new output is stored in the address that follows the previous output.

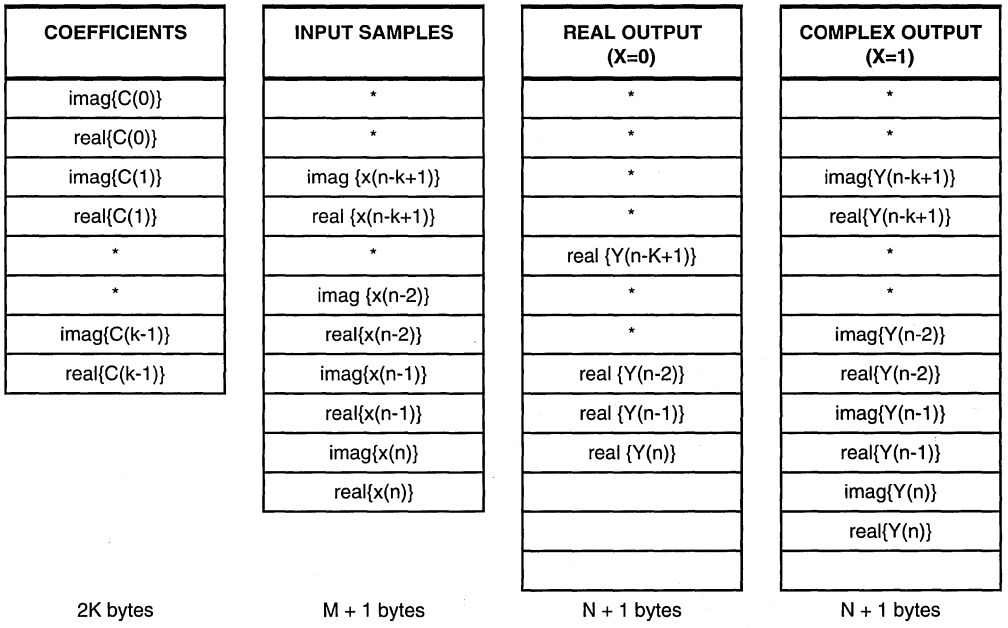


Figure 16-17. FIR3 Coefficients and Sample Data Buffers

16.3.4.3.2 FIR3 Function Descriptor. The FIR3 function descriptor bit table is described below.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	RES	W	I	Z	IALL	INDEX		PC	RES		OPCODE				
OFFSET + 2	I															
OFFSET + 4	K															
OFFSET + 6	CBASE															
OFFSET + 8	M															
OFFSET + A	XYPTR															
OFFSET + C	N															
OFFSET + E	RESERVED															

The first half-word is composed of the following bits:

S—STOP

0 = Do not stop after executing this function descriptor.
1 = Stop after executing this function descriptor.

Bits 1, 9, and 10—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Function Descriptor in Table)

0 = This is not the last function descriptor in the function descriptor table.
1 = This is the last function descriptor in the function descriptor table. After this buffer has been used, the CPM processes the first function descriptor that the FDBASE index pointer points to in the table. The number of function descriptors in this table are programmable and determined only by the W bit and overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

Z— Complex Output

0 = Only the real component of the result is written to the output buffer.
1 = The real and imaginary parts of the result are written to the output buffer.

IALL— Auto-Increment X For All Iterations

0 = The X (input) data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field after the last iteration.
1 = The X data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field after each iteration.

INDEX— Auto-Increment Index

00 = The X (input) pointer is not incremented.
01 = The X (input) pointer is incremented by one sample.
10 = The X (input) pointer is incremented by two samples.
11 = The X (input) pointer is incremented by three samples.

PC— Preset Coefficients Pointer

0 = The coefficients pointer is not preset after each iteration.
1 = The coefficients pointer is preset after each iteration to the CBASE pointer.

OPCODE—Function Operation Code

This field specifies the function to be executed. See Table 16-6 to get the value for this field.

16.3.4.3.3 FIR3 Parameter Packet. The FIR3 parameter packet is composed of seven 16-bit half-words and described in the table below.

Table 16-10. FIR3 Parameter Packet

ADDRESS	NAME	DESCRIPTION
Half-word 1	I	Number of Iterations
Half-word 2	K	Number of TAPs-1
Half-word 3	CBASE	Filter Coefficients Vector Base Address Pointer
Half-word 4	M	Samples Buffer Size-1. The minimum sample buffer size is 8 (2 samples).
Half-word 5	XYPTR	Pointer to a structure composed of the input sample data pointer and the output buffer pointer
Half-word 6	N	Output Buffer Size-1. The minimum output buffer size for x=1 is 8 (2 outputs). The minimum output buffer size for x=0 is 4 (2 outputs).
Half-word 7	RES	Reserved

16.3.4.3.4 Application Example. The FIR3 with the real output is used in echo cancellation and the one with the complex output is used in the equalizer.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	0	W	I	0	0	01		0	0	0	00011				
OFFSET + 2	I=3 (THREE ITERATIONS)															

16.3.4.4 FIR5–COMPLEX C, COMPLEX X, AND COMPLEX Y. The FIR5 function implements a basic FIR filter with k complex coefficients, complex input samples, and complex output. The input data is in a circular buffer with size M+1 and the output data is in a circular buffer with size N+1. The FIR5 only uses other input data samples to implement a fractionally spaced equalizer.

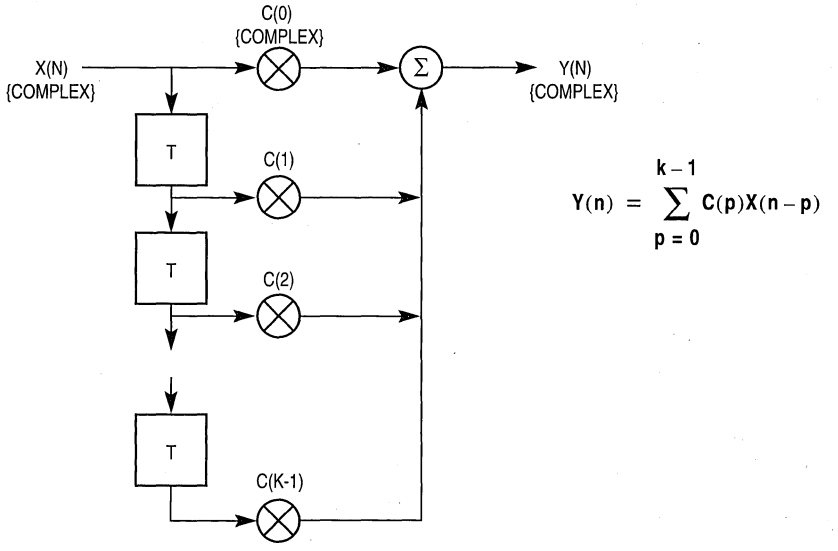


Figure 16-18. FIR5 Implementation Example

16.3.4.4.1 Coefficients and Sample Data Buffers. The coefficients vector occupies k pairs of 16-bit half-words (real and imaginary components) in memory and C(0) is stored in the first location. The sample input buffer is a cyclic buffer containing M+1 bytes. Each input sample is two 16-bit half-words (real and imaginary components) and the new sample is stored in the address that follows the previous sample. The output buffer is a cyclic buffer that contains N+1 bytes and the new output is stored in the address that follows the previous output.

COEFFICIENTS
imag{C(0)}
real{C(0)}
imag{C(1)}
real{C(1)}
*
*
imag{C(k-1)}
real{C(k-1)}

INPUT SAMPLES
*
*
imag {x(n-k+1)}
real {x(n-k+1)}
*
imag {x(n-2)}
real{x(n-2)}
imag{x(n-1)}
real{x(n-1)}
imag{x(n)}
real{x(n)}

REAL OUTPUT (X=0)
*
*
*
*
Y(n-K+1)
*
*
Y(n-2)
Y(n-1)
Y(n)

COMPLEX OUTPUT (X=1)
*
*
imag{Y(n-k+1)}
real{Y(n-k+1)}
*
*
imag{Y(n-2)}
real{Y(n-2)}
imag{Y(n-1)}
real{Y(n-1)}
imag{Y(n)}
real{Y(n)}

2K bytes M + 1 bytes N + 1 bytes N + 1 bytes

Figure 16-19. FIR5 Coefficients and Sample Data Buffers

16.3.4.4.2 FIR5 Function Descriptor. The FIR5 function descriptor bit table is described below.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	RES	W	I	Z	IALL	INDEX		PC	RES		OPCODE				
OFFSET + 2	I															
OFFSET + 4	K															
OFFSET + 6	CBASE															
OFFSET + 8	M															
OFFSET + A	XYPTR															
OFFSET + C	N															
OFFSET + E	RESERVED															

The first half-word is composed of the following bits:

S—STOP

0 = Do not stop after executing this function descriptor.
1 = Stop after executing this function descriptor.

Bits 1, 9, and 10—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Function Descriptor in Table)

0 = This is not the last function descriptor in the function descriptor table.
1 = This is the last function descriptor in the function descriptor table. After this buffer has been used, the CPM processes the first function descriptor that the FDBASE index pointer points to in the table. The number of function descriptors in this table are programmable and determined only by the W bit and overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

Z— Complex Output

0 = Only the real component of the result is written to the output buffer.
1 = The real and the imaginary parts of the result is written to the output buffer.

IALL— Auto-Increment X For All Iterations

0 = The X (input) data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field after the last iteration.
1 = The X data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field after each iteration.

INDEX— Auto-Increment Index

00 = The X (input) pointer is not incremented.
01 = The X (input) pointer is incremented by one sample.
10 = The X (input) pointer is incremented by two samples.
11 = The X (input) pointer is incremented by three samples.

PC— Preset Coefficients Pointer

0 = The coefficients pointer is not preset after each iteration.
1 = The coefficients pointer is preset after each iteration to the CBASE pointer.

OPCODE—Function Operation Code

This field specifies the function to be executed. See Table 16-6 to get the value for this field.

16.3.4.4.3 FIR5 Parameter Packet. The FIR5 parameter packet is composed of seven 16-bit half-words and described in the table below.

Table 16-11. FIR5 Parameter Packet

ADDRESS	NAME	DESCRIPTION
Half-word 1	I	Number of Iterations
Half-word 2	K	Number of TAPs-1.
Half-word 3	CBASE	Filter Coefficients Vector Base Address Pointer
Half-word 4	M	Samples Buffer Size-1. The minimum sample buffer size is 8 (2 samples).
Half-word 5	XYPTR	Pointer to a structure composed of the input sample data pointer and the output buffer pointer
Half-word 6	N	Output Buffer Size-1. The minimum output buffer size for x=1 is 8 (2 outputs). The minimum output buffer size for x=0 is 4 (2 outputs).
Half-word 7	RES	Reserved

16.3.4.4.4 Application Example. The FIR5 function is used in the fractionally spaced equalizer. The following example demonstrates how the function descriptor structure can be used to implement a fractionally spaced equalizer.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	0	W	I	1	0	11		0	0	0	00101				
OFFSET + 2	I=1 (ONE ITERATION)															

16.3.4.5 FIR6–COMPLEX C, REAL X, AND COMPLEX Y. The FIR6 function implements a basic FIR filter with k complex coefficients, real input samples, and complex output. The input data is in a circular buffer with size M+1 and the output data is in a circular buffer with size N+1.

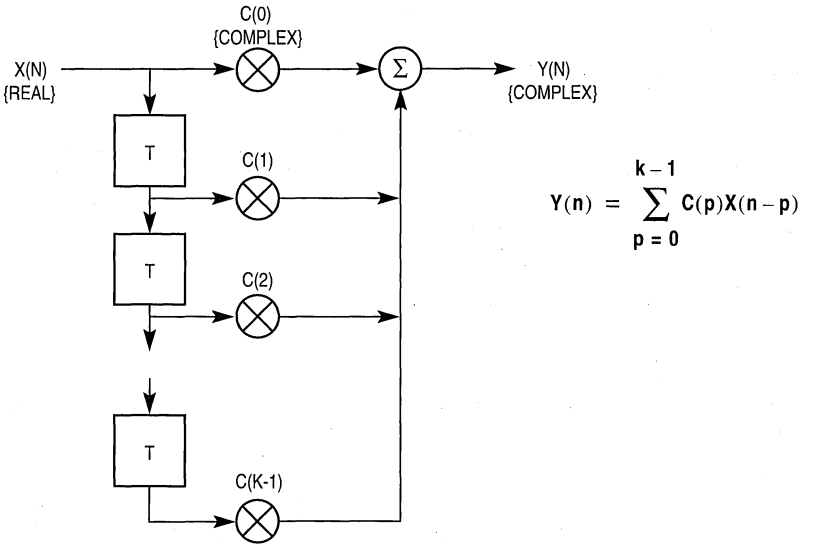


Figure 16-20. FIR6 Implementation Example

16.3.4.5.1 Coefficients and Sample Data Buffers. The coefficients vector occupies k pairs of 16-bit half-words (real and imaginary components) in memory and C(0) is stored in the first location. The sample input buffer is a cyclic buffer containing M+1 bytes and each sample is a 16-bit half-word. The new sample is stored in the address that follows the previous sample. The output buffer is a cyclic buffer that contains N+1 bytes and the new output is stored in the address that follows the previous output.

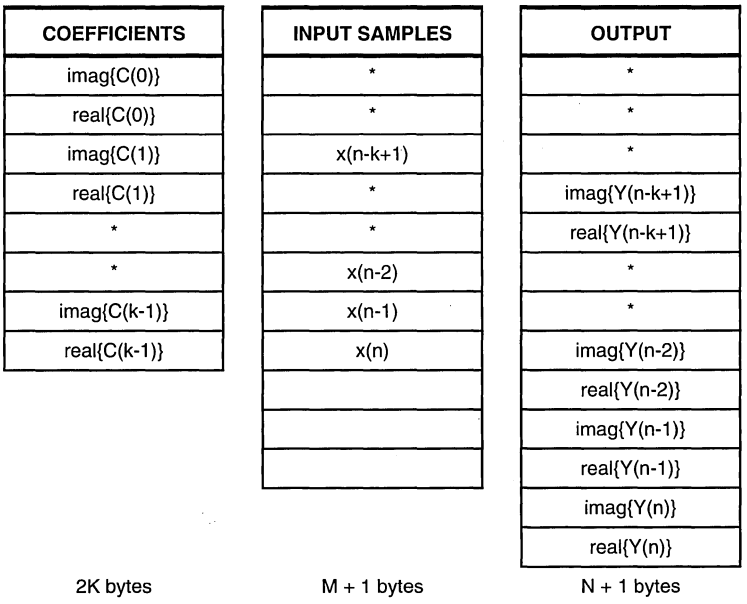


Figure 16-21. FIR6 Coefficients and Sample Data Buffers

16.3.4.5.2 FIR6 Function Descriptor. The FIR6 function descriptor bit table is described below.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	RES	W	I	RES	IALL	INDEX		PC	RES		OPCODE				
OFFSET + 2	I															
OFFSET + 4	K															
OFFSET + 6	CBASE															
OFFSET + 8	M															
OFFSET + A	XYPTR															
OFFSET + C	N															
OFFSET + E	RESERVED															

The first half-word is composed of the following bits:

S—STOP

0 = Do not stop after executing this function descriptor.
1 = Stop after executing this function descriptor.

Bits 1, 4, 9, and 10—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Function Descriptor in Table)

0 = This is not the last function descriptor in the function descriptor table.
1 = This is the last function descriptor in the function descriptor table. After this buffer has been used, the CPM processes the first function descriptor that the FDBASE index pointer points to in the table. The number of function descriptors in this table are programmable and determined only by the W bit and overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

IALL— Auto Increment X For All Iterations

0 = The X (input) data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field after the last iteration.
1 = The X data pointer is incremented (Modulo M+1) by the number of samples specified in the INDEX field after each iteration.

INDEX— Auto Increment Index

00 = The X (input) pointer is not incremented.
01 = The X (input) pointer is incremented by one sample.
10 = The X (input) pointer is incremented by two samples.
11 = The X (input) pointer is incremented by three samples.

PC— Preset Coefficients Pointer

0 = The coefficients pointer is not preset after each iteration.
1 = The coefficients pointer is preset after each iteration to the CBASE pointer.

OPCODE—Function Operation Code

This field specifies the function to be executed. See Table 16-6 to get the value for this field.

16.3.4.5.3 FIR6 Parameter Packet. The FIR6 parameter packet is composed of seven 16-bit half-words and described in the table below.

Table 16-12. FIR6 Parameter Packet

ADDRESS	NAME	DESCRIPTION
Half-word 1	I	Number of Iterations-1 (0 = one iteration)
Half-word 2	K	Number of TAPs-1. The number of taps should be a multiple of 2.
Half-word 3	CBASE	Filter Coefficients Vector Base Address Pointer
Half-word 4	M	Samples Buffer Size-1. The minimum sample buffer size is 4(2 samples).
Half-word 5	XYPTR	Pointer to a structure composed of the input sample data pointer and the output buffer pointer
Half-word 6	N	Output Buffer Size-1. The minimum output buffer size is 8 (2 outputs).
Half-word 7	RES	Reserved

16.3.4.6 IIR–REAL C, REAL X, REAL Y. The IIR function implements a basic BIQUAD IIR filter with six real coefficients, real input samples, and real outputs. The input data is in a circular buffer with size M+1 and the output data is in a circular buffer with size N+1. Several stages of the BIQUAD filter can be cascaded by specifying an iteration count greater than one and concatenating the filter coefficients into one vector.

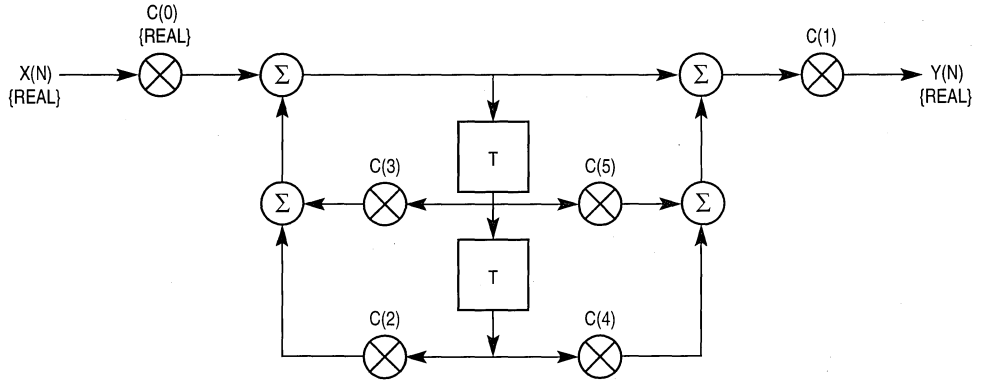


Figure 16-22. IIR Implementation Example

16.3.4.6.1 Coefficients and Sample Data Buffers. The coefficients vector occupies six 16-bit half-words in memory and C(0) is stored in the first location. C(1) is only used in the last stage of a cascaded IIR filter. The sample input buffer is a cyclic buffer that contains M+1 bytes. Each sample is a 16-bit half-word and the new sample is stored in the address that follows the previous sample. The output buffer is a cyclic buffer that contains N+1 bytes and the new output is stored in the address that follows the previous one.

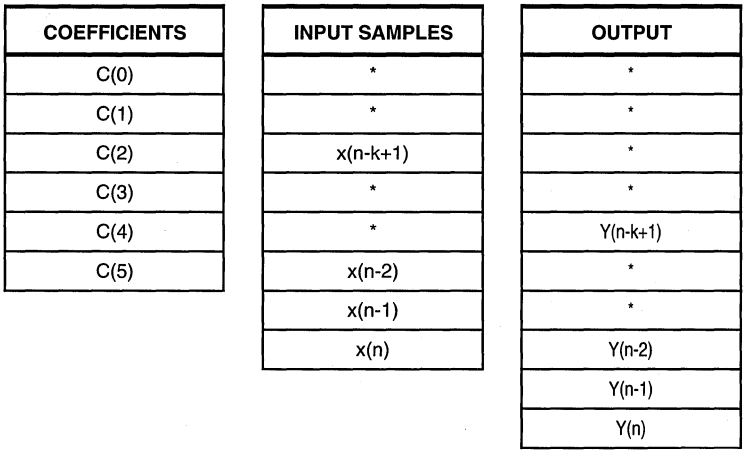


Figure 16-23. IIR Coefficients and Sample Data Buffers

16.3.4.6.2 IIR Function Descriptor. The IIR function descriptor bit table is described below.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	RES	W	I	RES		INDEX		RES			OPCODE				
OFFSET + 2	I															
OFFSET + 4	TPTR															
OFFSET + 6	CBASE															
OFFSET + 8	M															
OFFSET + A	XYPTR															
OFFSET + C	N															
OFFSET + E	RESERVED															

The first half-word is composed of the following bits:

S—STOP

0 = Do not stop after executiing this function descriptor.
1 = Stop after executing this function descriptor.

Bits 1, 4–5, and 8–10—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Function Descriptor in Table)

0 = This is not the last function descriptor in the function descriptor table.
1 = This is the last function descriptor in the function descriptor table. After this buffer has been used, the CPM processes the first function descriptor that the FDBASE index pointer points to in the table. The number of function descriptors in this table are programmable and determined only by the W bit and overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

INDEX— Auto-Increment Index

00 = The X (input) pointer is not incremented.
01 = The X (input) pointer is incremented by one sample.
10 = The X (input) pointer is incremented by two samples.
11 = The X (input) pointer is incremented by three samples.

OPCODE—Function Operation Code

This field specifies the function to be executed. See Table 16-6 to get the value for this field.

16.3.4.6.3 IIR Parameter Packet. The IIR parameter packet is composed of seven 16-bit half-words and described in the table below.

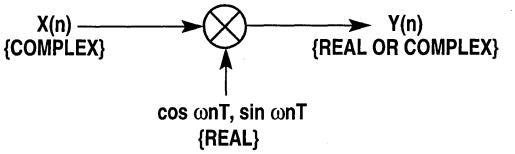
Table 16-13. IIR Parameter Packet

ADDRESS	NAME	DESCRIPTION
Half-word 1	I	Number of Iterations (= cascaded stages)
Half-word 2	TPTR	Pointer to temp delay line(s) pointer
Half-word 3	CBASE	Filter Coefficients Vector Base Address Pointer
Half-word 4	M	Samples Buffer Size-1. The minimum sample buffer size is 4 (2 samples).
Half-word 5	XYPTR	Pointer to a structure composed of the input sample data pointer and the output buffer pointer
Half-word 6	N	Output Buffer Size-1. The minimum output buffer size is 4 (2 outputs).
Half-word 7	RES	Reserved

16.3.4.6.4 Application Example. Among other things, the IIR is used in timing recovery and interpolating filter.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	0	W	I	0	0	INDEX		0	0	0	00111				
OFFSET + 2	I=1 (ONE ITERATION)															

16.3.4.7 MOD–REAL SIN, REAL COS, COMPLEX X, AND REAL/COMPLEX Y. The MOD function implements a basic modulator function with a modulation table composed of {cos ωnT, sin ωnT} pairs, complex input samples, and real outputs. The input data is in a circular buffer with size M+1 and the output data is in a circular buffer with size N+1.



$$\text{REAL}\{Y(n)\} = \text{REAL}\{X(n)\} \times \cos\omega nT - \text{IMAGE}\{X(n)\} \times \sin\omega nT$$
$$\text{IMAGE}\{Y(n)\} = \text{REAL}\{X(n)\} \times \sin\omega nT + \text{IMAGE}\{X(n)\} \times \cos\omega nT$$

Figure 16-24. MOD Implementation Example

16.3.4.7.1 Modulation Table and Sample Data Buffers. The modulation table is composed of 16-bit cosine and sine pairs that occupy K+1 bytes in memory. The sample input buffer is a cyclic buffer containing M+1 bytes. Each sample is a pair of 16-bit half-words (real and imaginary components) and the new sample is stored in the address that follows the previous sample. The output buffer is a cyclic buffer that contain N+1 bytes and the new output is stored in the address that follows the previous output. The output buffer can be real or complex, depending on the X bit in the function descriptor.

MODULATION TABLE
sin q_1
cos q_1
sin q_2
cos q_2
*
*
sin q_n
cos q_n

K + 1 bytes

INPUT SAMPLES
*
*
imag {x(n-k+1)}
real {x(n-k+1)}
*
imag {x(n-2)}
real{x(n-2)}
imag{x(n-1)}
real{x(n-1)}
imag{x(n)}
real{x(n)}

M + 1 bytes

OUTPUT (REAL)
*
*
*
real{Y(n-K+1)}
*
*
real{Y(n-2)}
real{Y(n-1)
real{Y(n)}

N + 1 bytes

OUTPUT (COMPLEX)
*
*
*
imag{Y(n-k+1)}
real{Y(n-k+1)}
*
*
imag{Y(n-2)}
real{Y(n-2)}
imag{Y(n-1)}
real{Y(n-1)}
imag{Y(n)}
real{Y(n)}

N + 1 bytes

Figure 16-25. MOD Table and Sample Data Buffers

16.3.4.7.2 MOD Function Descriptor. The MOD function descriptor bit table is described below.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	RES	W	I	Z	RES						OPCODE				
OFFSET + 2	I															
OFFSET + 4	K															
OFFSET + 6	MPTR															
OFFSET + 8	M															
OFFSET + A	XYPTR															
OFFSET + C	N															
OFFSET + E	RESERVED															

The first word is composed of the following bits:

S—STOP

0 = Do not stop after executing this function descriptor.
1 = Stop after executing this function descriptor.

Bits 1 and 5–10—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Function Descriptor in Table)

0 = This is not the last function descriptor in the function descriptor table.
1 = This is the last function descriptor in the function descriptor table. After this buffer has been used, the CPM processes the first function descriptor that the FDBASE index pointer points to in the table. The number of function descriptors in this table are programmable and determined only by the W bit and overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

Z— Complex Output

0 = Only the real component of the result is written to the output buffer.
1 = The real and imaginary parts of the result is written to the output buffer.

OPCODE—Function Operation Code

This field specifies the function to be executed. See Table 16-6 to get the value for this table.

16.3.4.7.3 MOD Parameter Packet. The MOD parameter packet is composed of seven 16-bit half-words and is described in the table below.

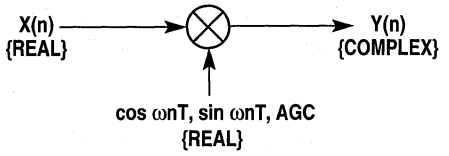
Table 16-14. MOD Parameter Packet

ADDRESS	NAME	DESCRIPTION
Half-word 1	I	Number of Iterations
Half-word 2	K	Modulation Table Size-1. The minimum modulation table size is 8 (2 sin/cos pairs).
Half-word 3	MPTR	Pointer to Modulation Table Pointer
Half-word 4	M	Samples Buffer Size-1. The minimum sample buffer size is 8 (2 samples).
Half-word 5	XYPTR	Pointer to a structure composed of the input sample data pointer and the output buffer pointer
Half-word 6	N	Output Buffer Size-1. The minimum output buffer size for x=1 is 8 (2 outputs). The minimum output buffer size for x=0 is 4 (2 samples).
Half-word 7	RES	Reserved

16.3.4.7.4 Application Example. The MOD is used in the modulator. The following example demonstrates how the function descriptor structure can be used to implement the MOD functions.

	0	1	2	3	4	5	6	7	8	9	0	11	12	13	14	15
OFFSET + 0	S	0	W	I	0	0	0	0	0	0	0	01000				
OFFSET + 2	I=3 (THREE ITERATIONS)															

16.3.4.8 DEMOD–REAL SIN; REAL COS, REAL X, AND COMPLEX Y. The DEMOD function implements a basic demodulator function with a modulation table composed of (cos ωnT, sin ωnT) pairs, real input samples, and complex outputs. The input data is in a circular buffer with size M+1 and the output data is in a circular buffer with size N+1. The AGC parameter controls the demodulator gain.



$$REAL\{Y(n)\} = (1 + AGC) \times X(n) \times \cos \omega nT$$

$$IMAGE\{Y(n)\} = (1 + AGC) \times X(n) \times (-\sin \omega nT)$$

Figure 16-26. DEMOD Implementation Example

16.3.4.8.1 Modulation Table, Sample Data Buffers, and AGC Constant. The modulation table is composed of 16-bit cosine and sine pairs that occupy K +1 bytes in memory. The samples input buffer is a cyclic buffer containing M+1 bytes. Each sample is a 16-bit half-word and the new sample is stored in the address that follows the previous sample. The output buffer is a cyclic buffer that contains N+1 bytes and the new output is stored in the address that follows the previous output. The AGC constant is in the range -1≤ AGC≤1.

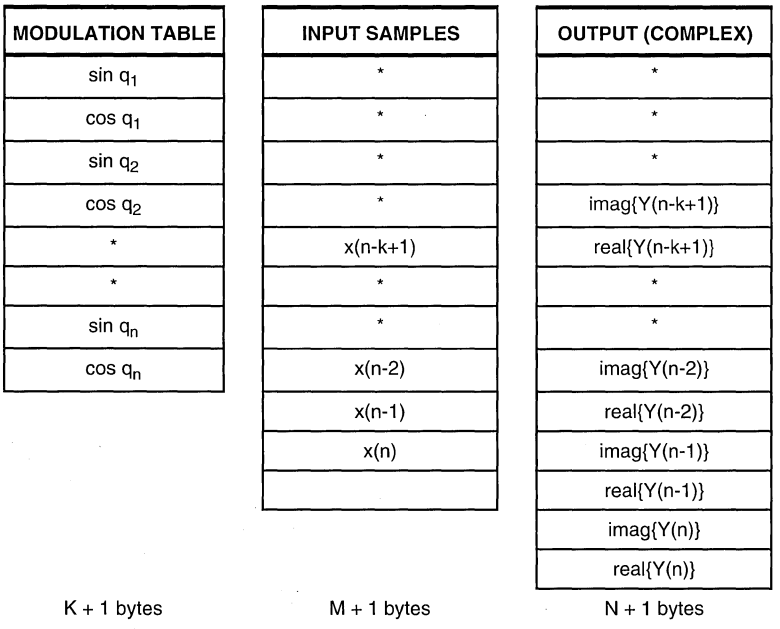


Figure 16-27. DEMOD Modulation Table and Sample Data Buffers

16.3.4.8.2 DEMOD Function Descriptor. The DEMOD function descriptor bit table is described below.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	RES	W	I	RES							OPCODE				
OFFSET + 2	I															
OFFSET + 4	K															
OFFSET + 6	DPTR															
OFFSET + 8	M															
OFFSET + A	XYPTR															
OFFSET + C	N															
OFFSET + E	RESERVED															

The first half-word is composed of the following bits:

S—STOP

0 = Do not stop after executing this function descriptor.
1 = Stop after executing this function descriptor.

Bits 1, 4–10—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Function Descriptor in Table)

0 = This is not the last function descriptor in the function descriptor table.
1 = This is the last function descriptor in the function descriptor table. After this buffer has been used, the CPM processes the first function descriptor that the FDBASE index pointer points to in the table. The number of function descriptors in this table are programmable and determined only by the W bit and overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

OPCODE—Function Operation Code

This field specifies the function to be executed. See Table 16-6 to get the value for this field.

16.3.4.8.3 DEMOD Parameter Packet. The DEMOD parameter packet is composed of seven 16-bit half-words and is described in the table below.

Table 16-15. DEMOD Parameter Packet

ADDRESS	NAME	DESCRIPTION
Half-word 1	I	Number of Iterations
Half-word 2	K	Modulation Table Size-1. The minimum modulation table size is 8 (2 sin/cos pairs).
Half-word 3	DPTR	Pointer to Modulation Table Pointer and AGC constant
Half-word 4	M	Samples Buffer Size-1. The minimum sample buffer size is 8 (2 samples).
Half-word 5	XYPTR	Pointer to a structure composed of the input sample data pointer and the output buffer pointer
Half-word 6	N	Output Buffer Size-1. The minimum output buffer size is 8 (2 outputs).
Half-word 7	RES	Reserved

16.3.4.8.4 Application Example. The DEMOD is used in the modulator. The following example demonstrates how the function descriptor structure can be used to implement the MOD function.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	0	W	I	0	0	0	0	0	0	0	01001				
OFFSET + 2	I=3 (THREE ITERATIONS)															

16.3.4.9 LMS1–COMPLEX COEFFICIENTS, COMPLEX SAMPLES, AND REAL/COMPLEX SCALAR . The LMS1 function implements a basic FIR filter coefficients update. The coefficients and input samples are complex numbers, but the scalar is a real or complex number.

$$C^{n+1}_i = C^n_i + E \times X_{n-i}$$

Figure 16-28. LMS1 Implementation Example

16.3.4.9.1 Coefficients and Sample Data Buffers. The coefficients vector occupies k pairs of 16-bit half-words (real and imaginary components) in memory and C(0) is stored in the first location. The samples input buffer is a cyclic buffer that contain M+1 bytes. Each sample is a pair of 16-bit half-words (real and imaginary components) and the new sample is stored in the address that follows the previous sample.

COEFFICIENTS
imag{C(0)}
real{C(0)}
imag{C(1)}
real{C(1)}
*
*
imag{C(k-1)}
real{C(k-1)}

2K bytes

INPUT SAMPLES
*
*
*
imag{X(n-k+1)}
real{X(n-k+1)}
*
*
imag{X(n-2)}
real{X(n-2)}
imag{X(n-1)}
real{X(n-1)}
imag{X(n)}
real{X(n)}

M + 1 bytes

Figure 16-29. LMS1 Coefficients and Sample Data Buffers

16.3.4.9.2 LMS1 Function Descriptor. The LMS1 function descriptor bit table is described below.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	RES	W	I	Z	RES	INDEX		RES			OPCODE				
OFFSET + 2	RESERVED															
OFFSET + 4	K															
OFFSET + 6	CBASE															
OFFSET + 8	M															
OFFSET + A	XYPTR															
OFFSET + C	EPTR															
OFFSET + E	RESERVED															

The first half-word is composed of the following bits:

S—STOP

0 = Do not stop after executing this function descriptor.
1 = Stop after executing this function descriptor.

W—Wrap (Final Function Descriptor in Table)

0 = This is not the last function descriptor in the function descriptor table.
1 = This is the last function descriptor in the function descriptor table. After this buffer has been used, the CPM processes the first function descriptor that the FDBASE index pointer points to in the table. The number of function descriptors in this table are programmable and determined only by the W bit and the overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

Z— Complex Scalar

0 = The scalar (E) is a real number.
1 = The scalar (E) is a complex number.

INDEX— Auto-Increment Index

00 = The X (input) pointer is not incremented.
01 = The X (input) pointer is incremented by one sample.
10 = The X (input) pointer is incremented by two samples.
11 = The X (input) pointer is incremented by three samples.

OPCODE—Function Operation Code

This field specifies the function to be executed. See Table 16-6 to get the value for this field.

16.3.4.9.3 LMS1 Parameter Packet. The LMS1 parameter packet is composed of seven 16-bit half-words and is described in the table below.

Table 16-16. LMS1 Parameter Packet

ADDRESS	NAME	DESCRIPTION
Half-word 1	RES	Reserved
Half-word 2	K	Number of Taps-1.
Half-word 3	CBASE	Filter Coefficients Vector Base Address Pointer
Half-word 4	M	Samples Buffer Size-1. The minimum sample buffer size is 8 (2 samples).
Half-word 5	XYPTR	Pointer to New Sample Data Pointer
Half-word 6	EPTR	Pointer to Scalar
Half-word 7	RES	Reserved

16.3.4.9.4 Application Example. The LMS1 is used in the echo cancellation update.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	0	W	I	Z	0	INDEX		0	0	0	01010				
OFFSET + 2	I=1 (ONE ITERATION)															

16.3.4.10 LMS2–COMPLEX COEFFICIENTS, COMPLEX SAMPLES, AND REAL/COMPLEX SCALAR. The LMS2 function implements a basic FIR filter coefficients update and the sample pointer is incremented by two, which is required for fractionally spaced equalizer updates. The coefficients and input samples are complex numbers, but the scalar is a real or complex number.

$$C^{n+1}_i = C^n_i + E \times X_{n-i}$$

Figure 16-30. LMS2 Implementation Example

16.3.4.10.1 Coefficients and Sample Data Buffers. The coefficients vector occupies k pairs of 16-bit half-words (real and imaginary components) in memory and C(0) is stored in the first location. The sample input buffer is a cyclic buffer containing M+1 bytes. Each sample is a pair of 16-bit half-words (real and imaginary components) and the new sample is stored in the address that follows the previous sample.

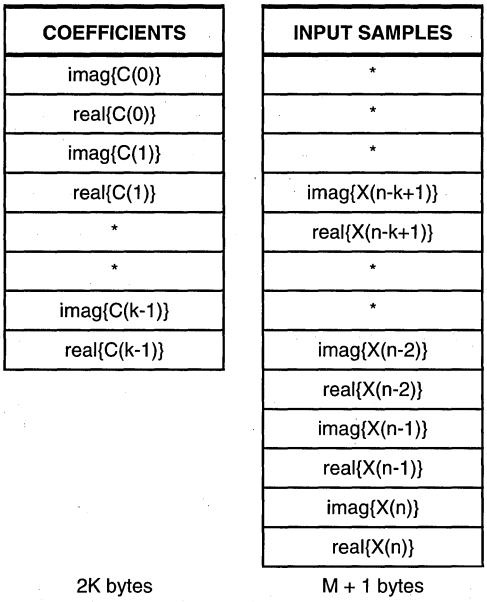


Figure 16-31. LMS2 Coefficients and Sample Data Buffers

16.3.4.10.2 LMS2 Function Descriptor. The LMS2 function descriptor bit table is described below.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	RES	W	I	Z	RES	INDEX		RES			OPCODE				
OFFSET + 2	RESERVED															
OFFSET + 4	K															
OFFSET + 6	CBASE															
OFFSET + 8	M															
OFFSET + A	XPTR															
OFFSET + C	EPTR															
OFFSET + E	RESERVED															

The first half-word is composed of the following bits:

S—STOP

0 = Do not stop after executing this function descriptor.
1 = Stop after executing this function descriptor.

Bits 1, 5, and 8–10—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Function Descriptor in Table)

0 = This is not the last function descriptor in the function descriptor table.
1 = This is the last function descriptor in the function descriptor table. After this buffer has been used, the CPM processes the first function descriptor that the FDBASE index pointer points to in the table. The number of function descriptors in this table are programmable and determined only by the W bit and overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

Z— Complex Scalar

0 = The scalar (E) is a real number.
1 = The scalar (E) is a complex number.

INDEX— Auto-Increment Index

00 = The X (input) pointer is not incremented.
01 = The X (input) pointer is incremented by one sample.
10 = The X (input) pointer is incremented by two samples.
11 = The X (input) pointer is incremented by three samples.

OPCODE—Function Operation Code

This field specifies the function to be executed. See Table 16-6 to get the value for this field.

16.3.4.10.3 LMS2 Parameter Packet. The LMS2 parameter packet is composed of seven 16-bit half-words and is described in the table below.

Table 16-17. LMS2 Parameter Packet

ADDRESS	NAME	DESCRIPTION
Half-word 1	RES	Reserved
Half-word 2	K	Number of Taps-1.
Half-word 3	CBASE	Filter Coefficients Vector Base Address Pointer
Half-word 4	M	Samples Buffer Size-1. The minimum sample buffer size is 8 (2 samples).
Half-word 5	XPTR	Pointer to New Sample Data Pointer
Half-word 6	EPTR	Pointer to Scalar
Half-word 7	RES	Reserved

16.3.4.10.4 Application Example. The LMS2 function is used in the fractionally spaced equalizer coefficient update.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	0	W	I	Z	0	INDEX		0	0	0	01011				
OFFSET + 2	I=1 (ONE ITERATION)															

16.3.4.11 WADD–REAL X AND REAL Y. The WADD function receives two real vectors and two real coefficients (α and β) as inputs. The function generates an output vector that is the linear combination between the two input vectors, according to α and β. It is a special case when $\beta = 1 - \alpha$ and $(0 \leq \alpha \leq 1)$ generates a linear interpolation between the two input vectors.

$$Y(n) = \alpha X_1(n) + \beta X_2(n)$$

Figure 16-32. WADD Implementation Example

16.3.4.11.1 Coefficients and Sample Data Buffers. Each input vector is stored in a cyclic buffer containing M+1 bytes. Each sample is a 16-bit half-word and the newest sample is stored in the address that follows the previous sample. The output buffer is a cyclic buffer that contains N+1 bytes. Each output is a 16-bit half-word and the newest output is stored in the address that follows the previous one.

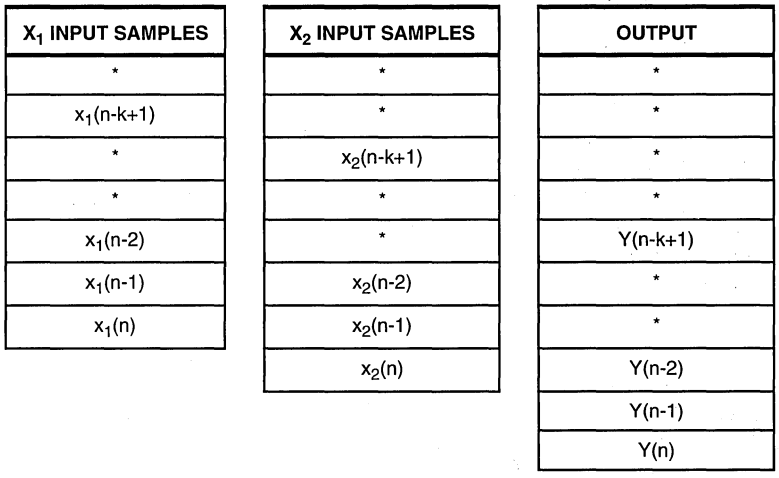


Figure 16-33. WADD Modulation Table and Sample Data Buffers

16.3.4.11.2 WADD Function Descriptor. The WADD function descriptor bit table is described below.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	S	RES	W	I	RES							OPCODE				
OFFSET + 2	I															
OFFSET + 4	A															
OFFSET + 6	B															
OFFSET + 8	M															
OFFSET + A	XYPTR															
OFFSET + C	N															
OFFSET + E	RESERVED															

The first half-word is composed of the following bits:

S—STOP

0 = Do not stop after executing this function descriptor.
1 = Stop after executing this function descriptor.

Bits 1 and 4–10—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Function Descriptor in Table)

0 = This is not the last function descriptor in the function descriptor table.
1 = This is the last function descriptor in the function descriptor table. After this buffer has been used, the CPM processes the first function descriptor that the FDBASE index pointer points to in the table. The number of function descriptors in this table are programmable and determined only by the W bit and overall space constraints of the memory.

I—Interrupt

0 = No interrupt is generated after this function is processed.
1 = A maskable interrupt is generated after this function is processed.

OPCODE—Function Operation Code

This field specifies the function to be executed. See Table 16-6 to get the value for this field.

16.3.4.11.3 WADD Parameter Packet. The WADD parameter packet is composed of seven 16-bit half-words and is described in the table below.

Table 16-18. WADD Parameter Packet

ADDRESS	NAME	DESCRIPTION
Half-word 1	I	Number of Iterations
Half-word 2	a	X_1 Weight Coefficient
Half-word 3	b	X_2 Weight Coefficient
Half-word 4	M	Samples Buffer Size-1
Half-word 5	XYPTR	Pointer to a structure composed of X_1 input sample data pointer, output buffer pointer, and the X_2 input sample data pointer.
Half-word 6	EPTR	Output Buffer Size-1
Half-word 7	RES	Reserved

16.3.4.11.4 Application Example. By specifying different values, several functions can be implemented as shown in the table below.

Table 16-19. WADD Functions

A	B	FUNCTION
$0 \leq \alpha \geq 1$	$1-\alpha$	Linear Interpolation
a	0	$y(n) = \alpha x(n)$ Scalar Multiply
1	-1	$y(n) = x_1(n) - x_2(n)$ Vector Subtract

16.3.4.12 THE DSP EXECUTION TIMES. A function's execution time is a linear function of the number of taps and iterations specified for that function. It includes overhead for context-switch, function descriptor handling, and initialization. Table 16-20 lists the execution time for each of the DSP functions.

Table 16-20. DSP Functions Execution Times

FUNCTION	EXECUTION TIME
FIR1	53 + 20 * (I - 1) + 1.25 * I * (K+1)
FIR2	47 + 17 * (I -1) + 3 * I * (K+1)
FIR3	44 + 14 * (I - 1) + 4 * I * (K+1)
FIR5	44 + 14 * (I - 1) + 5 * I * (K+1)
FIR6	50 + 20 * (I - 1) + 3 * I * (K+1)
IIR	44 + 11 * I
MOD	44 + 7 * I
DEMOD	47 + 14 * I
LMS1	42 + 7 * (K+1)
LMS2	42 + 7 * (K+1)
WADD	46 + 7 * I

NOTES:

1. Add 1 clock for wrap, 5 clocks for stop, and 4 clocks for interrupt.
2. i = number of iterations.
3. k+1 = number of taps.

16.4 TIMERS

The communication processor module includes four identical 16-bit general-purpose timers that can be cascaded into two 32-bit timers. Each general-purpose timer consists of a timer mode register, a timer capture register, a timer counter, a timer reference register, a timer event register, and a timer global configuration register. The timer mode register contains the prescaler value that you program. The timer block diagram is illustrated in Figure 16-34.

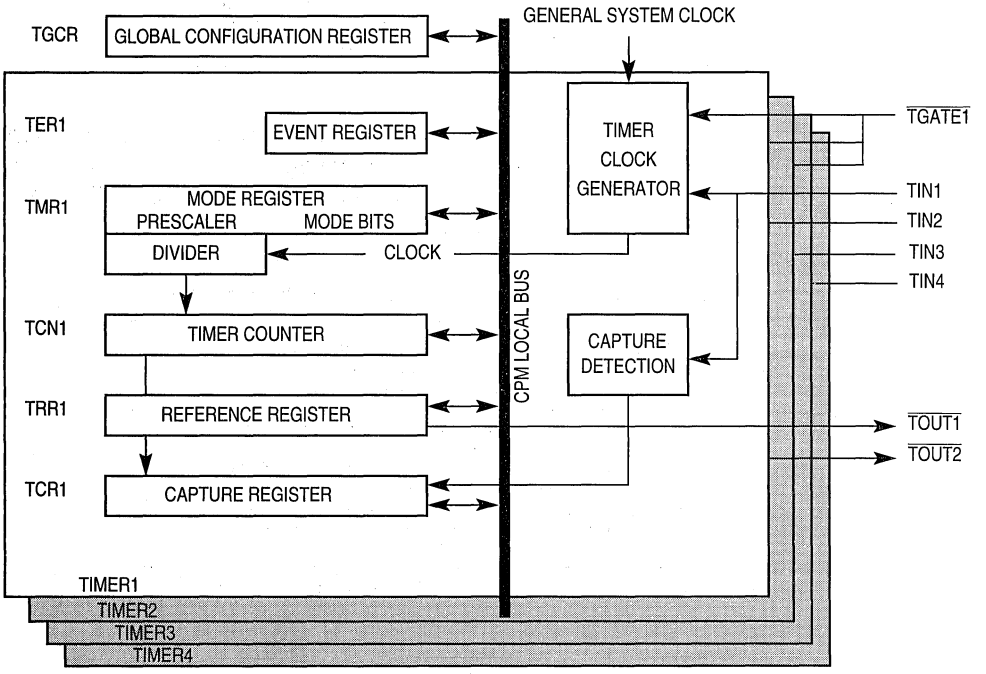


Figure 16-34. Timer Block Diagram

16.4.1 Features

The following list summarizes the main features of the timers:

- Maximum period of 10.7 seconds (at 25MHz)
- Minimum 40ns resolution (at 25MHz)
- Programmable sources for the clock input
- Input capture capability
- Output compare with programmable mode for the output pin
- Four timers can be internally or externally cascaded to form two 32-bit timers. However, if you are using Mask Revision Base #F98S, there are only two timers.
- Free run and restart modes

16.4.2 Timer Operation

The clock input to the prescaler can be selected from three sources:

- The general system clock
- The general system clock divided by 16
- The corresponding TIN*x* pin

The general system clock is generated in the clock synthesizer and defaults to the system frequency (25 or 50MHz). However, the general system clock has the option to be divided before it leaves the clock synthesizer. This mode, called normal low, is used to save power. Whatever the resulting frequency of the general system clock, you can either choose that frequency or the frequency divided by 16 as the input to the prescaler of each timer. On the other hand, you may prefer that the TINx pin be the clock source because it is internally synchronized to the internal clock.

The clock input source is selected by the ICLK field of the corresponding timer mode register (TMRx). The prescaler is programmed to divide the clock input by values between 1 and 256 and the output of the prescaler is used as an input to the 16-bit counter. The best resolution of the timer is one clock cycle (40ns at 25MHz). The maximum period is 268,435,456 cycles, which is 10.7 seconds at 25MHz. Both values assume that the general system clock is the full 25MHz.

Each timer can be configured to count until a reference is reached and then either begin a new time count immediately or continue running. The FRR bit of the corresponding TMRx selects each mode. When the reference value is reached, the corresponding TER bit is set and an interrupt is issued if the ORI bit in the TMRx is set. Timers 1 and 2 can output a signal on the timer output pin ($\overline{\text{TOUT1}}$ and $\overline{\text{TOUT2}}$) when the reference value is reached or selected by the OM bit of the corresponding TMR1 and TMR2. This signal can be an active-low pulse or a toggle of the current output.

In addition, each timer has a 16-bit timer capture register (TCRx) that is used to latch the value of the counter when a defined transition of the TIN1, TIN2, TIN3, or TIN4 pin is sensed by the corresponding input capture edge detector. The type of transition triggering the capture is selected by the CE field in the corresponding TMRx. When a capture or reference event occurs, the corresponding TER bit is set and a maskable interrupt request is issued to the CPM interrupt controller. Timers 1 and 2 can be gated or restarted using the $\overline{\text{TGATE1}}$ signal (timers 3 and 4 cannot be gated). Normal gate mode enables the count on the falling edge of the $\overline{\text{TGATE1}}$ pin and disables the count on the rising edge of the $\overline{\text{TGATE1}}$ pin. This mode allows the timer to count conditionally, depending on the state of the $\overline{\text{TGATE1}}$ pin.

Restart gate mode performs the same function as normal mode, except it also resets the counter on the falling edge of the $\overline{\text{TGATE1}}$ pin. This mode can be used in pulse interval measurement and bus monitoring applications:

- Pulse Measurement—The restart gate mode can measure a low pulse on the $\overline{\text{TGATE1}}$ pin. The rising edge of the $\overline{\text{TGATE1}}$ pin completes the measurement and if $\overline{\text{TGATE1}}$ is externally connected to TINx, it causes the timer to capture the count value and generate a rising-edge interrupt.
- Bus Monitoring—The restart gate mode can detect a signal that is abnormally stuck low. The bus signal should be connected to the $\overline{\text{TGATE1}}$ pin. The timer count is reset on the falling edge of the bus signal and if the bus signal does not go high again within the number of user-defined clocks, an interrupt can be generated.

The gate function is enabled in the timer mode register and the gate operating mode is selected in the timer global configuration register.

Note: $\overline{\text{TGATE1}}$ is internally synchronized to the timebase clock (TMBCLK). If it meets the asynchronous input setup time, then (when working with the internal clock) the counter begins counting after one system clock.

16.4.2.1 CASCADED MODE. In this mode, the 16-bit timers can be internally cascaded into a 32-bit counter. Since the decision to cascade timers is made independently, you have the option of selecting four 16-bit timers or two 32-bit timers. The timer global configuration register is used to put the timers into cascaded mode. See Figure 16-35 for details.

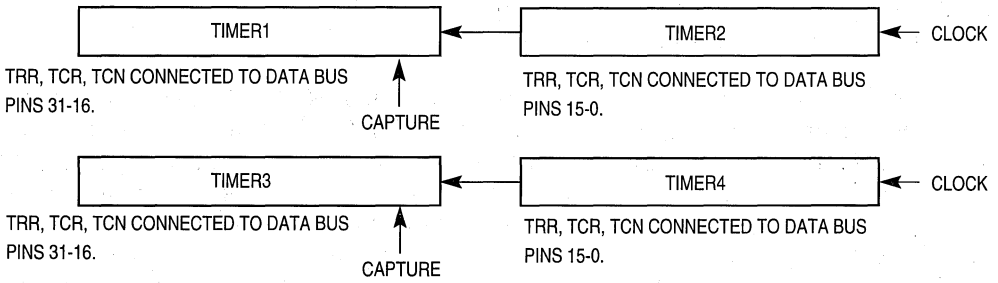


Figure 16-35. Timer Cascaded Mode Block Diagram

If the CAS2 bit is set in the timer global configuration register, the two timers function as a 32-bit timer with a 32-bit timer reference register, timer capture register, and timer counter. In this case, timers 1 and 3 are ignored and timers 2 and 4 should be used to define the mode. The capture is controlled by the TIN2 pin and the interrupts are generated by the TER2 pin. When operating in cascaded mode, the cascaded timer reference register, timer capture register, and timer counter should always be referenced with 32-bit bus cycles.

16.4.2.2 TIMER GLOBAL CONFIGURATION REGISTER. The 16-bit, memory-mapped, read/write timer global configuration register (TGCR) contains configuration parameters that are used by both timers. It allows simultaneous starting and stopping of any number of timers as long as one bus cycle is used to access TGCR.

TGCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	CAS4	FRZ4	STP4	RST4	RES	FRZ3	STP3	RST3	CAS2	FRZ2	STP2	RST2	GM1	FRZ1	STP1	RST1
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x980															

CAS4—Cascade Timers

0 = Normal operation.
1 = Timers 3 and 4 are cascaded to form a 32-bit timer.

FRZ4–FRZ1—Freeze

0 = The corresponding timer ignores the FRZ pin.
1 = Stops the corresponding timer if the FRZ pin is asserted by the core during breakpoint.

STP4–STP1 —Stop Timer

0 = Normal operation.
1 = Reduce the timer's power consumption. This bit stops all clocks to the timer, except the clock from the U-Bus interface, which allows you to read and write the timer registers. The clocks to the timer remain inactive until you clear this bit or a hardware reset occurs.

RST4–RST1—Reset Timer

0 = Reset the corresponding timer. A software reset is identical to an external reset.
1 = Enable the corresponding timer if the STPx bit is cleared.

Bit 4—Reserved

This bit is reserved and should be set to 0.

CAS2—Cascade Timers

0 = Normal operation.
1 = Timers 1 and 2 are cascaded to form a 32-bit timer.

GM1—Gate Mode for Pin 1

This bit is only valid if the gate function is enabled in timer 1 or 2.

0 = Restart gate mode. A falling $\overline{\text{TGATE1}}$ pin enables and restarts the count and a rising edge of $\overline{\text{TGATE1}}$ disables the count.
1 = Normal gate mode. This mode is the same as 0, except the falling edge of the $\overline{\text{TGATE1}}$ pin does not restart the count value in the timer counter.

16.4.2.3 TIMER MODE REGISTERS. The 16-bit, memory-mapped, read/write timer mode registers (TMR1–TMR4) are identical and cleared by reset. To avoid erratic behavior, the TGCR must be initialized before the TMRx. The only exception is that the RST bit in the TGCR can be modified at any time.

TMR1–TMR4

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PS								CE		OM	ORI	FRR	ICLK		GE
RESET	0								0		0	0	0	0		0
R/W	R/W								R/W		R/W	R/W	R/W	R/W		R/W
ADDR	(IMMR & 0xFFFF0000) + 0x990 (TMR1), 0x992 (TMR2), 0x9A0 (TMR3), 0x9A2 (TMR4)															

PS—Prescaler Value

The prescaler is programmed to divide the clock input by a value between 1 and 256. A 00000000 value divides the clock by 1 and 11111111 divides it by 256.

CE—Capture Edge and Enable Interrupt

00 = Disable interrupt on capture event; capture function is disabled.
01 = Capture on rising TINx edge only and enable interrupt on capture event.
10 = Capture on falling TINx edge only and enable interrupt on capture event.
11 = Capture on any TINx edge and enable interrupt on capture event.

OM—Output Mode (Only Valid for TMR1 and TMR2)

0 = Active-low pulse on $\overline{\text{TOUT1}}$ or $\overline{\text{TOUT2}}$ for one timer input clock cycle as defined by the ICLK bits. Thus, $\overline{\text{TOUTx}}$ may be low for one general system clock period, one general system clock/16 period, or one TINx pin clock cycle period. Changes to $\overline{\text{TOUTx}}$ occur on the rising edge of the system clock.
1 = Toggle the $\overline{\text{TOUTx}}$ pin. Changes to $\overline{\text{TOUTx}}$ occur on the rising edge of the system clock.

ORI—Output Reference Interrupt Enable

0 = Disable interrupt for reference that is reached. This does not affect an interrupt on the capture function.
1 = Enable interrupt when the reference value is reached.

FRR—Free Run/Restart

0 = Free run. The timer count continues to increment after the reference value is reached.
1 = Restart. The timer count is reset immediately after the reference value is reached.

ICLK—Input Clock Source for the Timer

00 = Internally cascaded input.
For TMR1, the timer 1 input is the output of timer 2.
For TMR3, the timer 3 input is the output of timer 4.
For TMR2 and TMR4, this selection means no input clock is provided to the timer.
01 = Internal general system clock.
10 = Internal general system clock divided by 16.
11 = Corresponding TINx pin (falling edge).

GE—Gate Enable

0 = The $\overline{\text{TGATE1}}$ signal is ignored.
1 = The $\overline{\text{TGATE1}}$ signal is used to control the timer.

16.4.2.4 TIMER REFERENCE REGISTERS. Each of the 16-bit, memory-mapped, read/write timer reference registers (TRR1–TRR4) contain the timeout's reference value.

TRR1–TRR4

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	REFERENCE															
RESET	1															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x994 (TRR1), 0x996 (TRR2), 0x9A4 (TRR3), 0x9A6 (TRR4)															

REFERENCE—Reference to TCNx

This reference value is reached when the TCNx register increments to equal TRRx.

16.4.2.5 TIMER CAPTURE REGISTERS. Each of the 16-bit, memory-mapped, read-only timer capture registers (TCR1–TCR4) are used to latch the value of the counter at the falling edge of every TINx signal. The CE field in the TMRx defines whether the counter starts at the rising or falling edge of the TINx signal.

TCR1–TCR4

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	CAPTURE COUNT															
RESET	0															
R/W	R															
ADDR	(IMMR & 0xFFFF0000) + 0x998 (TCR1), 0x99A (TCR2), 0x9A8 (TCR3), 0x9AA (TCR4)															

CAPTURE COUNT

This field contains the value of the timer counter register at the falling edge of the TINx signal.

16.4.2.6 TIMER COUNTER REGISTERS. Each of the 16-bit, memory-mapped, read/write timer counter (TCN1 and TCN4) registers are up-counters. In timer capture mode, the TINx signal defines when the counter begins. A read to TCNx yields the current value of the timer, but does not affect the counting operation. A write to TCNx sets the register to that value, which causes the corresponding prescaler in the PS field of the TMRx to be reset.

TCN1–TCN4

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	COUNT															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x99C (TCN1), 0x99E (TCN2), 0x9AC (TCN3), 0x9AE (TCN4)															

NOTE: — = Undefined.

COUNT—Counter Value

This represents the value of the counter.

Note: The TCNx register may not be updated correctly if a write is made to it while the timer is not running. You should always use the TRRx to define the preferred counter value.

16.4.2.7 TIMER EVENT REGISTERS. Each of the 16-bit, memory-mapped timer event registers (TER1–TER4) are used to report events recognized by the timers. When an output reference event is recognized, the timer sets the REF bit, regardless of the corresponding ORI bit in the TMRx. The capture event is only set if it is enabled by the CE field in the TMRx. These registers can be read at any time and are cleared at reset. A bit is reset by writing a 1 (writing a zero has no effect) and more than one bit can be reset at a time. However, both bits must be reset before the timer negates the interrupt to the CPM interrupt controller.

TER1–TER4

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED														REF	CAP
RESET	0														0	0
ADDR	(IMMR & 0xFFFF0000) + 0x9B0 (TER1), 0x9B2 (TER2), 0x9B4 (TER3), 0x9B6 (TER4)															

Bits 0–13—Reserved

These bits are reserved and should be set to 0.

REF—Output Reference Event

This bit signifies that the counter has reached the value in the TRRx. The ORI bit in the TMRx is used to enable the interrupt request caused by this event.

CAP—Capture Event

This bit signifies that the counter value has been latched into the TCRx. The CE field in the TMRx is used to enable generation of this event.

16.4.3 Initializing the Timers

The following is an example of timer 2's initialization sequence that is required to generate an interrupt every 10 microseconds, assuming a 25MHz general system clock. An interrupt should be generated after each 250 system clock.

1. Write 0x0000 to the TGCR. This puts timer 2 into a reset state. Cascaded mode is not used here.
2. Write 0x001A to the TMR2. This enables the timer's prescaler to divide by 1 and sets the clock source to general system clock. It also enables an interrupt to occur when the reference value is reached and restarts the timer to continuously generate 10ms interrupts.
3. Write 0x0000 to the TCN2 register. This initializes the timer 2 count to zero (default state of this register).
4. Write 0x00FA to the TRR2. This initializes the timer 2 reference value to 250 (decimal).

5. Write 0xFFFF to the TER2. This clears the TER2 of any bits that might have been set.
6. Write 0x00040000 to the CIMR. This enables a timer 2 interrupt in the CPM interrupt controller and initializes the CICR.
7. Write 0x0010 to the TGCR. This enables timer 2 to begin counting.

To implement the same function with a 32-bit timer using timers 1 and 2, follow these steps:

1. Write 0x0080 to the TGCR. This cascades timers 1 and 2 and puts them in a reset state.
2. Write 0x001A to the TMR2. This enables the timer 2 prescaler to divide by 1 and the clock source to the general system clock. It also enables an interrupt when the reference value is reached and restarts the timer to continuously generate 10ms interrupts.
3. Write 0x0000 to the TMR1. This enables timer 1 to use the output of timer 2 as its input (default of TMR1).
4. Write 0x0000 to the TCN1 and 0x0000 to the TCN2. This initializes the count of the combined timers 1 and 2 to zero (default of TMR1) by using one 32-bit data move to TCN1.
5. Write 0x0000 to the TRR1 and 0x00FA to the TRR2. This initializes the reference value of the combined timers 1 and 2 to 250 by using one 32-bit data move to TRR1.
6. Write 0xFFFF to the TER2. This clears the TER2 of any bits that might have been set.
7. Write 0x00040000 to the CIMR. This enables the timer 2 interrupt in the CPM interrupt controller and initializes the CICR.
8. Write 0x0091 to the TGCR. This enables timers 1 and 2 to begin counting, but leaves them in cascaded mode.

16.5 THE SDMA CHANNELS

The MPC823 has two physical serial DMA (SDMA) channels. One is controlled by the RISC microcontroller and the other by the LCD controller. The RISC microcontroller implements 12 virtual SDMA channels and each one is associated with a serial channel transmitter or receiver. Four channels are associated with the full-duplex serial communication controller and the other eight are used for the serial peripheral interface, I^2C controller, and serial management controllers. Each channel is permanently assigned to service either the receive or transmit operation of a serial communication controller, serial management controller, serial peripheral interface, or I^2C controller. Data from these controllers can be routed to the external RAM (path 1) or the internal dual-port RAM (path 2) with the U-Bus. Figure 16-36 illustrates the paths of the data flow.

On a path 1 access, the U-Bus and external system bus must be acquired by the SDMA channel. On a path 2 access, only the U-Bus needs to be acquired and the access is not seen on the external system bus, unless the MPC823 is configured into the "show cycles" mode of the system interface unit. Thus, transfers on the U-Bus occur at the same time that other operations on the external system bus occur.

Each SDMA channel can be programmed to output one of eight function codes that identify the channel currently accessing memory. The SDMA channel can be assigned a big-endian (Motorola) or little-endian format for accessing buffer data. These features are programmed in the receive and transmit function code registers that are associated with the serial communication controller, serial management controllers, serial peripheral interface, and I^2C controller. If a bus error occurs when the SDMA conducts a RISC-related access, the communication processor module generates a unique interrupt in the RISC status register.

The interrupt service routine reads the SDMA address register to determine the address that the bus error occurred on. The channel that caused the bus error can be found by reading the RX and TX internal data pointers from the specific protocol parameters area in the parameter RAM of the serial channels. If an SDMA bus error occurs on a RISC-related cycle, all CPM activity ceases and the entire communication processor module must be reset in the CPM command register (CPCR).

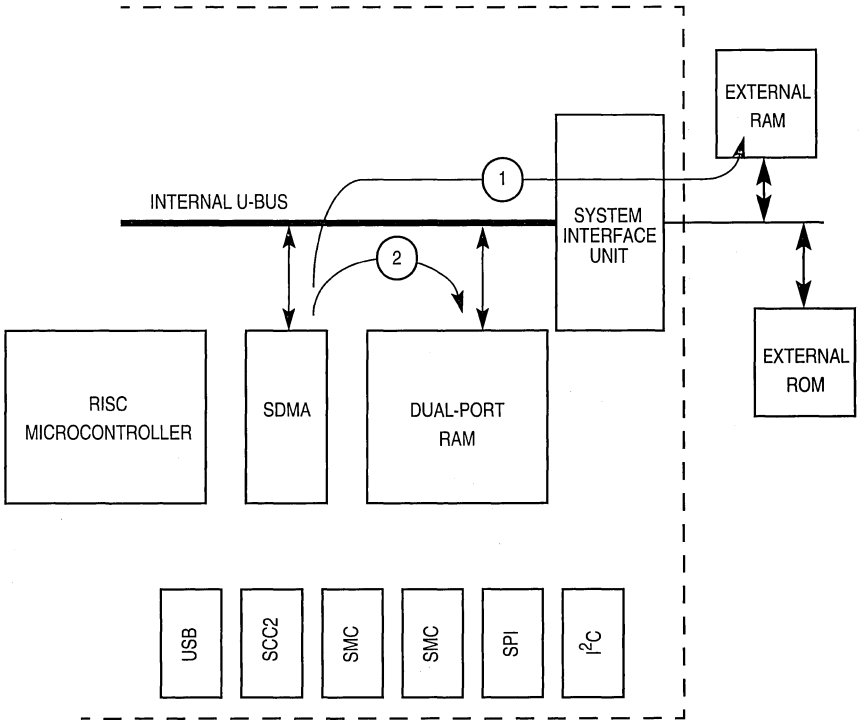


Figure 16-36. SDMA Data Paths

16.5.1 SDMA Bus Arbitration and Transfers

The instruction cache, data cache, system interface unit, and SDMA can become internal bus masters whose relative priority can be determined by examining their arbitration IDs. However, you can only adjust SDMA arbitration. All other arbitration IDs are fixed. All 12 SDMA channels share the same ID that you are responsible for programming. Any SDMA channel can arbitrate for the bus against the other internal or external masters that are present. Once an SDMA channel obtains the system bus, it remains the bus master for one transaction (a byte, half-word, word, or burst) before relinquishing the bus. This feature, in combination with the zero clock arbitration overhead provided by the U-Bus, makes a simultaneous benefit out of bus efficiency and low bus latency.

With character-oriented protocols, the SDMA writes characters to memory without waiting for multiple characters to be received first and it always reads words. This is consistent with the need to provide low-latency operation on character-oriented protocols that are used at slower rates. The read or write operation may take multiple bus cycles if the memory provides a less than 32-bit port size. For instance, a 32-bit word read from a 16-bit memory takes two SDMA bus cycles. The entire operand (4-word burst, 32 bits on reads, and 8, 16, or 32 bits on writes) will be transferred in back-to-back bus cycles before the SDMA relinquishes the bus. The SDMA can steal cycles with no arbitration overhead when the MPC823 is bus master.

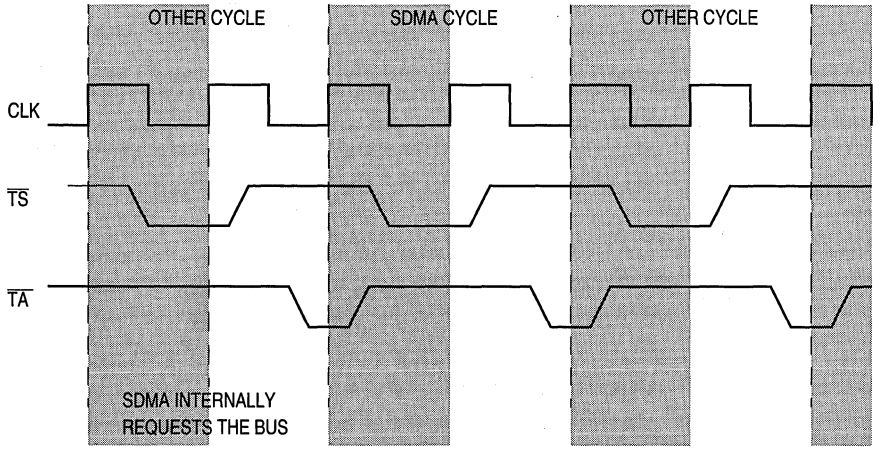


Figure 16-37. SDMA Bus Arbitration

16.5.2 The SDMA Registers

The SDMA channels share a configuration register, address register, and status register. They are all controlled by the configuration of the serial communication controller, serial management controllers, serial peripheral interface, and I^2C controller.

16.5.2.1 SDMA CONFIGURATION REGISTER. The 32-bit SDMA configuration register (SDCR) is used to configure all 16 SDMA channels. It is always read/write in supervisor mode, even though writing to the SDCR is not recommended unless the communication processor module is disabled. The control provided by this register has interactions with the DMA controllers in the LCD and video controller modules of the MPC823. Refer to **Section 18.3.6 DMA Control** and **Section 18.3.1 FIFO Control** for more information regarding those modules.

SDCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
RESET	0															
R/W	R															
ADDR	(IMMR & 0xFFFF0000) + 0x030															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RES	FRZ		RESERVED						LAM	RESERVED		LAID		RAID	
RESET	0	0		0						0	0		0		0	
R/W	R	R		R						R/W	R		R		R	
ADDR	(IMMR & 0xFFFF0000) + 0x030															

Bits 0–16—Reserved

These bits are reserved and should be set to 0.

FRZ—Freeze

These bits determine the action to be taken when the FRZ signal is asserted. The SDMA negates the $\overline{BR}$ signal and keeps it that way until FRZ is negated or a reset occurs.

00 = The SDMA channels ignore the FRZ signal.
01 = Reserved.
10 = The SDMA channels freeze on the next bus cycle.
11 = Reserved.

LAM—LCD/Video Aggressive Mode

0 = Disable LCD aggressive mode. Priority depends on the LAID field.
1 = Enable LCD aggressive mode. The LAID and RAID fields must be equal to 00.

Bits 19–24 and 26–27—Reserved

These bits are reserved and should be set to 0.

LAID—LCD Controller Arbitration ID

This field determines the LCD and video controller arbitration ID. Its value should be programmed to 00 for typical applications.

00 = The LCD controller uses U-Bus arbitration priority 6 (highest).
01 = The LCD controller uses U-Bus arbitration priority 5.
10 = The LCD controller uses U-Bus arbitration priority 2.
11 = The LCD controller uses U-Bus arbitration priority 1 (lowest).

RAID—RISC Controller Arbitration ID

These bits establish the priority level of bus arbitration among modules that can become bus master. The instruction cache, data cache, system interface unit, and SDMAs all compete for bus mastership. The SDMA channel arbitration ID is determined by the RAID field. Arbitration IDs for all other bus masters are internally fixed. This value should be programmed to 01 for typical applications.

00 = The SDMA uses U-Bus arbitration priority 6 (highest).
01 = The SDMA uses U-Bus arbitration priority 5.
10 = The SDMA uses U-Bus arbitration priority 2.
11 = The SDMA uses U-Bus arbitration priority 1 (lowest).

16.5.2.2 SDMA STATUS REGISTER. Shared by all 12 SDMA channels, the 8-bit memory-mapped SDMA status register (SDSR) is used to report events recognized by the SDMA controller. When an event is recognized, the SDMA sets the corresponding bit in the SDSR. A bit is reset by writing a 1 (writing a zero has no effect) and more than one bit can be reset at a time. This register is cleared by reset and can be read at any time.

SDSR

BIT	0	1	2	3	4	5	6	7
FIELD	SBER	RESERVED					DSP2	DSP1
RESET	0	0					0	0
R/W	R/W	R/W					R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x908							

SBER—SDMA Channel Bus Error (SDMA function)

This bit indicates that an error caused the SDMA channel to terminated during a read or write cycle. The SDMA bus error address can be read from the SDMA address register.

Bits 1–5—Reserved

These bits are reserved and should be set to 0.

DSP2—DSP Chain 2 Transmitter Interrupt (DSP function)

This bit is set when the chain 2 function finishes executing. However, the I bit must be set in the function descriptor, as shown in **Section 16.3.3.3 DSP Event Register**.

DSP1—DSP Chain 1 Receiver Interrupt (DSP function)

This bit is set when the chain 1 function finishes executing. However, the I bit must be set in the function descriptor, as shown in **Section 16.3.3.3 DSP Event Register**.

16.5.2.3 SDMA MASK REGISTER. The 8-bit read/write SDMA mask register (SDMR) has the same bit format as the SDMA status register. If a bit in the SDMA mask register is a 1, the corresponding interrupt in the event register is enabled. If the bit is zero, the corresponding interrupt is masked.

SDMR

BIT	0	1	2	3	4	5	6	7
FIELD	SBER	RESERVED					DSP2	DSP1
RESET	0	0					0	0
R/W	R/W	R/W					R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x90C							

SBER—SDMA Channel Bus Error Mask (SDMA function)

0 = Disable the SDMA channel bus error interrupt.
1 = Enable the SDMA channel bus error interrupt.

Bits 1–5—Reserved

These bits are reserved and should be set to 0.

DSP2—DSP Chain 2 Transmitter Interrupt Mask (DSP function)

0 = Disable the DSP chain 2 transmitter interrupt, as shown in **Section 16.3.3.3 DSP Event Register**.
1 = Enable the DSP chain 2 transmitter interrupt.

DSP1—DSP Chain 1 Receiver Interrupt Mask (DSP function)

0 = Disable the DSP chain 1 receiver interrupt, as shown in **Section 16.3.3.3 DSP Event Register**.
1 = Enable the DSP chain 1 receiver interrupt.

16.5.2.4 SDMA ADDRESS REGISTER. The 32-bit read-only SDMA address register (SDAR) shows the system address that is accessed during an SDMA bus error. It is undefined at reset.

SDAR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	SBEA															
RESET	—															
R/W	R															
ADDR	(IMMR & 0xFFFF0000) + 0x904															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	SBEA															
RESET	—															
R/W	R															
ADDR	(IMMR & 0xFFFF0000) + 0x906															

NOTE: — = Undefined.

SBEA—SDMA Bus Error Address

This field contains the system address accessed when the SDMA encounters a bus error.

16.6 EMULATING IDMA

Using the IDMA-dedicated SDMA channel, which is not part of the 12 channels, the RISC microcontroller can be configured to provide a general-purpose DMA functionality. Two general-purpose independent DMA (IDMA) channels are supported. In this special emulation mode, you can specify any memory-to-memory or peripheral-to-memory transfers to be implemented in the same way as dedicated DMA hardware.

The general-purpose IDMA controllers operate in different data transfer modes that you are responsible for programming. They can transfer data between any combination of memory and I/O. In addition, data can be transferred in byte, half-word, word, or burst quantities and the source and destination addresses can be odd or even. The most efficient packing algorithms are used in IDMA transfers. The single address mode performs the best because it allows data to be transferred between memory and the peripheral within a single bus cycle. The chip-select and wait-state generation logic on the MPC823 can be used with IDMA.

IDMA supports three buffer handling modes—single buffer, autobuffer, and buffer chaining. The single buffer mode allows a single buffer peripheral to memory data transfer with single address (flyby) burst transfers. The autobuffer mode allows blocks of data to be repeatedly moved from one location to another without you having to intervene. The buffer chaining mode allows a chain of blocks to be moved sequentially. You specify how to move the data using buffer descriptors similar to those used by a serial communication controller. These buffer descriptions reside in the dual-port RAM.

16.6.1 Features

The following is a list of the MPC823 IDMA's main features:

- Two independent, fully programmable DMA channels
- Dual address or single address transfers with 32-bit address and data capability
- 32-bit byte transfer counters
- 32-bit address pointers that can increment or remain constant
- Efficient operand packing and unpacking for dual address transfers
- All bus-termination modes are supported
- Provides DMA handshake for cycle steal and burst transfers
- Buffer handling modes (auto buffer and buffer chaining)

16.6.2 IDMA Interface Signals

The MPC823 IDMA has two dedicated control signals per channel—DMA request and DMA acknowledge. IDMA accepts DMA requests from the $\overline{\text{DREQ1}}$ and $\overline{\text{DREQ2}}$ signals and acknowledges the request with the $\overline{\text{SDACK1}}$ and $\overline{\text{SDACK2}}$ signals. The peripheral used with these signals can either be a source or destination of the IDMA transfers. The $\overline{\text{DREQx}}$ signals are also used for memory-to-memory request generation and, in this case, should be connected to the timer that controls the transfer.

16.6.2.1 $\overline{\text{DREQx}}$ AND $\overline{\text{SDACKx}}$. These are the handshake signals between the MPC823 and the peripheral that needs to be serviced. When the peripheral asks for IDMA service, it asserts $\overline{\text{DREQx}}$ and the MPC823 begins the IDMA process. While the service is in progress, $\overline{\text{SDACKx}}$ is asserted during accesses to the device. $\overline{\text{DREQx}}$ can be configured to be either edge or level sensitive by programming the DRxM field in the RCCR. The DRQP field in the RCCR control IDMA channel priority in relation to the serial channels. To enable the $\overline{\text{DREQx}}$ signals, the corresponding DREQx bit in the PCSO register should be set. When the $\overline{\text{DREQx}}$ signals are configured as edge-sensitive requests, the edge on which a request is generated is controlled by the corresponding EDMx bit in the PCINT register. See **Section 16.14.6 The Port B Registers** for details on the Port C registers.

16.6.3 IDMA Operation

Every IDMA operation involves the following series of events—IDMA channel initialization, data transfer, and block termination. In the initialization phase, the core loads the IDMA-specific parameter RAM with control information, initializes the IDMA buffer descriptors, and starts the channel. In the transfer phase, IDMA accepts requests for operand transfers and provides addressing and bus control for the transfers. The termination phase occurs when the operation is complete and IDMA interrupts the core if interrupts are enabled. To initialize a block transfer operation, you must initialize the IDMA registers. The IDMA buffer descriptors must be initialized with information describing the data block, device type, and other special control options. Refer to **Section 16.6.3.2 IDMA Parameter RAM Memory Map** and **Section 16.6.3.6 IDMA Commands** for more details.

Follow these steps to perform an IDMA transfer:

1. Define the source (peripheral) address to be burst aligned.
2. Define the destination address to be burst aligned.
3. Program the IDMA mode register (DCMR) to 0x0000.

16.6.3.1 AUTOBUFFER AND BUFFER CHAINING. The host CPU should initialize the IDMA buffer descriptor ring with the appropriate buffer handling mode, source address, destination address, and block length.

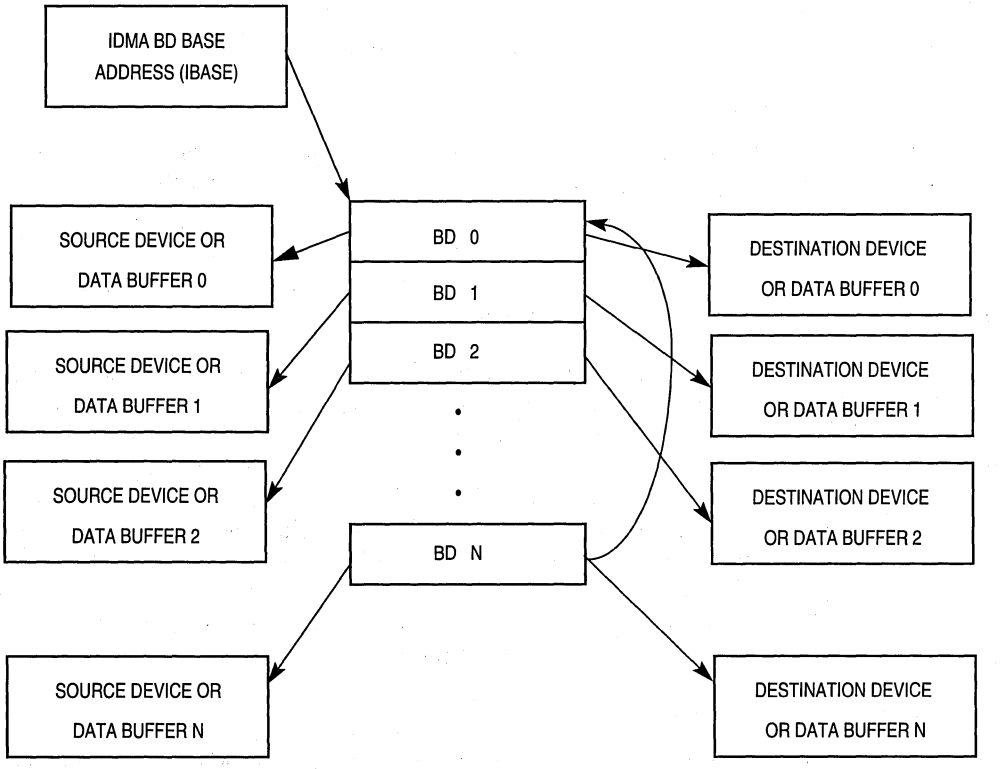


Figure 16-38. IDMA Buffer Descriptor Ring

The data associated with each IDMA channel for autobuffer and buffer chaining modes is stored in buffers and each buffer is referenced by a buffer descriptor that uses a ring structure located in the dual-port RAM.

16.6.3.2 IDMA PARAMETER RAM MEMORY MAP. The MPC823 uses the IDMA parameters listed in the table below to configure the IDMA channel for autobuffer or buffer chaining mode.

Table 16-21. IDMA Parameter RAM Memory Map

ADDRESS	NAME	WIDTH	DESCRIPTION
IDMA Base + 00	**IBASE**	Half-word	IDMA BD Base Address Index Pointer
IDMA Base + 02	**DCMR**	Half-word	IDMA Channel Mode Register
IDMA Base + 04	SAPR	Word	Source Internal Address Pointer
IDMA Base + 08	DAPR	Word	Destination Internal Address Pointer
IDMA Base + 0C	IBPTR	Half-word	Buffer Descriptor Pointer
IDMA_Base +0E	WRITE_SP	Half-word	—
IDMA Base + 10	S_BYTE_C	Word	Internal Source Byte Count
IDMA Base + 14	D_BYTE_C	Word	Internal Destination Byte Count
IDMA Base + 18	S_STATE	Word	Internal State
IDMA Base + 1C	ITEMP	Four-word	Temporary Data Storage
IDMA Base + 2C	SR_MEM	Word	Data Storage for Peripheral Write
IDMA Base + 30	READ_SP	Half-word	—
IDMA Base + 32	—	Half-word	Diff Between Source and Destination Residue
IDMA Base + 34	—	Half-word	Temporary Storage Address Pointer
IDMA Base + 36	—	Half-word	SR_MEM Byte Count
IDMA Base + 38	D_STATE	Word	Internal State

NOTE: You are only responsible for initializing the items in bold.
IDMA Base = (IMMR & 0xFFFF0000) + 0x3CC0 (IDMA1) or 0x3DC0 (IDMA2).
All references to registers in the parameter RAM table are actually implemented in the dual-port RAM area as a memory-based register.

- IBASE—This index pointer defines the starting point in the dual-port RAM for the set of IDMA buffer descriptors. It is an offset from the beginning of the dual-port RAM. You must initialize this entry before enabling the IDMA channel and if you overlap the buffer descriptor tables of two enabled serial channels or IDMA channels, erratic behavior will occur. IBASE should contain a value that is divisible by 16.

- DCMR—This register controls the operation mode of the IDMA channel.

DCMR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED											SIZE		TYPE		SC
RESET	0											0		0		0
R/W	R/W											R/W		R/W		R/W
ADDR	(IMMR & 0xFFFF0000) + 0x3CC2 (IDMA1) AND 0x3CD2 (IDMA2)															

Bits 0–10—Reserved
These bits are reserved and should be set to 0.

SIZE—Peripheral Port Size
00 = Word length.
01 = Half-word length.
10 = Byte length.
11 = Reserved.

TYPE—Source/Destination Type (Can Be Peripheral or Memory)
00 = Read and write from memory.
01 = Read from peripheral and write to memory.
10 = Read from memory and write to peripheral.
11 = Reserved.

SC—Single Cycle
0= Dual-cycle mode.
1= Single-cycle mode.

The following bits are only used by the RISC microcontroller:

- SAPR—This address pointer points to the next source data byte that IDMA transfers.
- DAPR—This address pointer points to the next destination byte that IDMA writes. When the buffer descriptor is first processed, the communication processor module initializes these pointers to the programmed values in the buffer descriptor.
- IBPTR—This pointer points to the next buffer descriptor that the IDMA transfers data to when it is in idle state or points to the current buffer descriptor during transfer processing. After a reset or when the end of an IDMA buffer descriptor table is reached, the communication processor module initializes this pointer to the value programmed in the IBASE register.
- WRITE_SP—This parameter should not be modified.
- S_BYTE_C—This parameter should not be modified.
- D_BYTE_C—This parameter should not be modified.

- S_STATE—This parameter should not be modified.
- ITEMP—This parameter should not be modified.
- SR_MEM—This parameter should not be modified.
- READ_SP—This parameter should not be modified.
- D_STATE—This parameter should not be modified.

The source address pointer points to the next source data byte that the IDMA transfers. The destination address pointer points to the next destination byte that the IDMA writes. When the buffer descriptor is first being processed, the CPM initializes these pointers to the programmed values. The remaining parameters are to only be used by the microcontroller. The memory port size is transparent to the IDMA, regardless of the actual port size. The system interface unit emulates 32-bit port size.

16.6.3.3 IDMA STATUS REGISTER. The 8-bit, memory-mapped IDMA status register (IDSR) is used to report events recognized by the IDMA controller. When an event is recognized, IDMA sets its corresponding bit in this register. A bit is reset by writing a 1 (writing a zero has no effect) and more than one bit can be reset at a time. This register is cleared by reset and can be read at any time.

IDSR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED					AD	DONE	OB
RESET	0					0	0	0
R/W	R/W					R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x910 (IDSR1), 0x918 (IDSR2)							

Bits 0–4—Reserved

These bits are reserved and should be set to 0.

AD—After Service Buffer Descriptor Done

This status bit is set after servicing a buffer descriptor that has the I bit set.

DONE—IDMA Transfer Done

This bit indicates when the IDMA channel terminates a transfer. It is set after servicing a buffer descriptor that has the L status bit set.

OB—Out of Buffers

This bit indicates that the IDMA channel has no valid buffer descriptors.

16.6.3.4 IDMA MASK REGISTER. The 8-bit read/write IDMA mask register (IDMR) has a different bit format than the IDSR. If a bit in the IDMR is a 1, the corresponding interrupt in the status register is enabled and if it is zero, the corresponding interrupt in the status register is masked. The OB and AD bits are in a different sequence from the corresponding bits in the IDSR.

IDMR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED					OB	DONE	AD
RESET	0					0	0	0
R/W	R/W					R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x914 (IDMR1), 0x91C (IDMR2)							

Bits 0–4—Reserved

These bits are reserved and should be set to 0.

OB—Out of Buffers

This bit indicates that the IDMA channel has no valid buffer descriptors.

DONE—IDMA Transfer Done

This bit indicates when the IDMA channel terminates a transfer. It is set after servicing a buffer descriptor that has the L status bit set.

AD—After Service Buffer Descriptor Done

This status bit is set after servicing a buffer descriptor that has the I bit set.

16.6.3.5 IDMA BUFFER DESCRIPTORS. The special IDMA buffer descriptors present the source addresses, destination addresses, and byte counts to the RISC microcontroller. The RISC microcontroller reads the buffer descriptors, programs the SDMA channel, and notifies the core about the completion of a buffer transfer using the IDMA buffer descriptors. This process is similar to that of the serial channels, except that the buffer descriptor is larger because it contains additional information.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	V	RES	W	I	L	RES	CM	RESERVED								
OFFSET + 2	DFCR							SFCR								
OFFSET + 4	DATA LENGTH															
OFFSET + 6																
OFFSET + 8	SOURCE DATA BUFFER POINTER															
OFFSET + A																
OFFSET + C	DESTINATION DATA BUFFER POINTER															
OFFSET + E																

NOTE: You are only responsible for initializing the items in bold.

You must prepare the following bits before a transfer can occur. They are set by the RISC microcontroller after the buffer has been transferred.

V—Valid

0 = The data buffers associated with this buffer descriptor are not currently ready for transfer. You are free to manipulate this buffer descriptor or its associated data buffer. When it is not in autobuffer mode, the RISC microcontroller clears this bit after the buffer has been transferred (or after an error condition is encountered).

1 = You have prepared the data buffers for transfer. Notice that only one data buffer should be prepared if the source/destination is a peripheral device. It can only be the source data buffer when the destination is a device or the destination data buffer when the source is a device. You can not write any fields of this buffer descriptor once this bit is set.

Note: The only difference between autobuffer mode and buffer chaining mode is that the V bit is not cleared by the RISC microcontroller in autobuffer mode. Autobuffer mode is enabled by the CM bit.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the table.

1 = This is the last buffer descriptor in the table. After the associated buffer has been used, the RISC microcontroller transfers data from the first buffer descriptor that IBASE register points to in the table. The number of buffer descriptors in this table is programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is serviced.

1 = Once this buffer is serviced by the RISC microcontroller, the AD bit in the IDSR is set and can cause an interrupt.

L—Last

0 = This is not the last buffer to be transferred in the buffer chaining mode. The I bit can be used to generate an interrupt when this buffer is serviced.

1 = This is the last buffer to be transferred in the buffer chaining mode. When the transfer count is exhausted, an interrupt is generated, regardless of state of the I bit.

CM—Continuous Mode

0 = Buffer chaining mode. The RISC microcontroller clears the V bit after this buffer descriptor is serviced. This mode is used to transfer large amounts of data into noncontiguous buffer areas. You can initialize buffer descriptors ahead of time, if you need to. The RISC microcontroller automatically reloads the IDMA registers from the next buffer descriptor values when the transfer is terminated.

1 = Autobuffer mode (continuous mode). The RISC microcontroller does not clear the V bit after this buffer descriptor is serviced. This is the only difference between the behavior of autobuffer mode and buffer chaining mode. The autobuffer mode is used to transfer multiple groups of data to/from a buffer ring and does not require reprogramming. The RISC microcontroller automatically reloads the IDMA registers from the next buffer descriptor values when the transfer is terminated. Either a single buffer descriptor or multiple buffer descriptors can be used in this mode to create an infinite loop of repetitive data moves.

Note: The I bit can be used to generate an interrupt in this mode.

SFCR—Source Function Code Register

The 8-bit source function code register contains the value that you would like to appear on the AT pins when the associated DMA channel accesses the source memory. This register controls the byte-ordering convention used in transfers.

SFCR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED			BO		AT1	AT2	AT3
ADDR	OFFSET + 3							

Bits 0–2—Reserved

These bits are reserved and should be set to 0.

BO—Byte Ordering

You should set this field to select the required byte ordering of the data buffer. If this field is modified on-the-fly, it takes effect at the beginning of the next frame or at the beginning of the next buffer descriptor.

00 = The DEC/Intel convention is used for byte ordering (swapped operation). It is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode can only be used with 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation) is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT—Address Type 1–3

This field contains the function code value used during the SDMA channel memory access. AT0 is always driven to 1 so that this SDMA channel access is identified as a DMA-type access.

DFCR—Destination Function Code Register

The 8-bit destination function code register contains the value that you would like to appear on the AT pins when the associated DMA channel accesses the destination memory. This register also controls the byte-ordering convention used in transfers.

DFCR

BITS	0	1	2	3	4	5	6	7
FIELD	RESERVED			BO		AT1	AT2	AT3
ADDR	OFFSET + 2							

Bits 0–2—Reserved

These bits are reserved and should be set to 0.

BO—Byte Ordering

You should set this field to select the data buffer's required byte ordering. If these bits are modified on-the-fly, it takes effect at the beginning of the next frame or at the beginning of the next buffer descriptor.

00 = The DEC/Intel convention is used for byte ordering (swapped operation). It is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode can only be used with 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation) is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT1–AT3—Address Type 1–3

This field contains the function code value used during SDMA channel memory access. AT0 is driven with a 1 so that this SDMA channel access is identified as a DMA-type access.

DATA LENGTH

This field contains the number of bytes that IDMA should transfer to or from this buffer descriptor data buffer. It should be programmed to a value greater than zero.

SOURCE BUFFER POINTER

This field contains the address of the associated source data buffer and can reside in internal or external memory.

Note: In single address mode when the source is a device, the SOURCE BUFFER POINTER field is ignored. In dual address mode when the source is a device, this field should contain the device address.

DESTINATION BUFFER POINTER

This field contains the address of the associated destination data buffer, which resides in internal or external memory.

Note: In single address mode when the destination is a device, the DESTINATION BUFFER POINTER field is ignored. In dual address mode when the destination is a device, this field should contain the device address.

The terminal count code AT = 0xF replaces the normal SFCR and DFCR code for the last IDMA cycle on the peripheral side.

16.6.3.6 IDMA COMMANDS. There are three IDMA commands:

- **INIT_IDMA**—This command causes the RISC microcontroller to reinstall its IDMA internal state to the condition it was in after a system reset. The IDMA buffer descriptor pointer is reinitialized to the top of the buffer descriptor ring.
- **ARM_IDMA**—This command causes the IDMA to open the next buffer descriptor in the table. It can be used to reduce the latency of servicing the first IDMA request.
- **STOP_IDMA**—This command causes the RISC microcontroller to terminate current IDMA transfers. The DONE bit is set in the IDSR and the current buffer descriptor is closed. If the peripheral device is the source, the IDMA internal buffer is transferred to memory before termination. At the next request, the following buffer descriptor in the chain is processed.

16.6.3.7 STARTING IDMA. Once the channel has been initialized with all parameters required for a transfer operation, IDMA is started by setting the DREQx bit in the port C special option (PCSO) register. Once DREQx has been set, the channel is active and accepts operand transfer requests through the channel's corresponding $\overline{\text{DREQx}}$ pin. When the first valid external request is recognized, IDMA arbitrates for the bus and the $\overline{\text{DREQx}}$ pin input is ignored until the DREQx bit is set in the PCSO register. IDMA1 is higher than IDMA2.

The software can suspend channel transfer operation at any time by clearing the DREQx bit in the PCSO register. In response, any operand transfer in progress will be completed and the bus will be released. No further bus cycles are started while the DREQx bit remains cleared. During this time, the core can access IDMA internal registers to determine the status of a channel or to alter operation. When the DREQx bit is set again, if a transfer request is pending, IDMA arbitrates for the bus and continues normal operation. Interrupts from IDMA are sent to the interrupt controller. In the interrupt handler, the unmasked bits in the IDSR should be cleared (by writing them with a 1) to negate the interrupt request to the CPM interrupt controller.

16.6.3.8 REQUESTING IDMA TRANSFERS. Once IDMA has been started, transfers to IDMA can be requested. These transfers are initiated by externally generated requests from an external device using the $\overline{\text{DREQx}}$ pin in conjunction with the activation of the DREQx bit of the PCSO register.

16.6.3.9 LEVEL-SENSITIVE MODE. For external devices that require very high data transfer rates, level-sensitive mode allows IDMA to use a maximum bandwidth to service the device. In this mode, the $\overline{\text{DREQx}}$ pin input to IDMA is level-sensitive and sampled at rising edges of the clock to determine when a valid request is asserted by the device. The device requests service by asserting the $\overline{\text{DREQx}}$ pin and leaving it asserted as long as it needs service.

Each time IDMA issues a bus cycle to either read or write the device, it outputs the $\overline{\text{SDACKx}}$ signal. The device is either the source or destination of the transfers, as determined by the TYPE field of the DCMR. $\overline{\text{SDACKx}}$ is the acknowledgment of the original burst request given on the $\overline{\text{DREQx}}$ pin. $\overline{\text{DREQx}}$ should be negated during the $\overline{\text{SDACKx}}$ active period to guarantee that no further cycles are performed. Burst mode in the context of the $\overline{\text{DREQx}}$ pin actually means back-to-back DMA cycles. Burst in the context of the bus means burst cycle. The DMA always uses bursts at the memory width.

16.6.3.10 EDGE-SENSITIVE MODE. For external devices that generate a pulsed signal for each operand to be transferred, edge-sensitive mode should be used. In edge-sensitive mode, IDMA moves one operand for each falling edge of the $\overline{\text{DREQx}}$ pin input. $\overline{\text{DREQx}}$ is sampled at each rising edge of the clock to determine when a valid request is asserted by the device. When IDMA detects a falling edge on the $\overline{\text{DREQx}}$ pin, a request becomes pending and remains as such until it is serviced by IDMA. Further falling edges on the $\overline{\text{DREQx}}$ pin are ignored until the request starts being serviced, which results in one operand being transferred. Each time IDMA issues a bus cycle to read or write the device, it outputs the $\overline{\text{SDACKx}}$ signal. The device is either the source or destination of the transfers, as determined by the TYPE field of the DCMR. Thus, $\overline{\text{SDACKx}}$ is the acknowledgment of the original cycle steal request given on the $\overline{\text{DREQx}}$ pin.

16.6.3.11 IDMA OPERAND TRANSFERS. Once IDMA successfully arbitrates for the bus, it can begin making operand transfers. The source IDMA bus cycle has timing identical to an internal master read bus cycle. The destination IDMA bus cycle's timing is controlled by the memory controller and is therefore identical to any other internal access. The two-channel IDMA controller supports dual-address transfers and single address transfers, but only channel 1 supports single-buffer (single-address burst fly-by) transfers. The dual-address operand transfer consists of a source operand read and a destination operand write. Each single-address operand transfer consists of one external bus cycle that allows a read or write cycle to occur. A single-buffer mode transfer is exactly like the single-address operand transfer, except it is an external burst transfer.

16.6.3.11.1 Transfer Identification. The following are ways to externally determine if IDMA is executing a bus cycle:

- Monitor the AT signals or SDMA channels to determine if they have the unique function code that identifies an IDMA transfer.
- Monitor the $\overline{\text{SDACKx}}$ signal, which shows accesses to the peripheral device. $\overline{\text{SDACKx}}$ activates on either the source or destination bus cycles, depending on the TYPE field of the DCMR.

16.6.3.11.2 Dual-Address Mode. The two IDMA channels can be programmed to operate in a dual-address transfer mode in which the operand is read from the source address specified by the pointer and placed in internal storage. The operand read can take several bus cycles to complete because of differences in source and destination operand sizes. Once it is read, the operand is then written to the address specified in the destination address pointer. This transfer may also be several bus cycles long. You can use a variety of peripheral, memory, and operand size combinations in a dual-address mode transfer. There are two types of dual-address mode cycles:

- Dual-address source read—Initially, the IDMA controller copies the source data buffer pointer buffer descriptor into the SAPR field of the parameter RAM. During this type of IDMA cycle, the SDBP is used to drive the address bus, the SFCR is used to drive the source address type, and the DCMR is used to drive the size control. Data is read from the memory or peripheral and placed in the internal storage when the bus cycle is terminated. When the complete operand has been read, the SAPR is incremented by 1, 2, 4, or 16, depending on the address and size information specified by the IDMA channel mode register (DCMR).
- Dual-address destination write—Initially, the IDMA controller copies the destination data buffer pointer buffer descriptor into the DAPR field of the parameter RAM. During this type of IDMA cycle, the data in the internal storage is written to the device or memory selected by the AT field in the DAPR, the AT field in the DFCR, and the SIZE field in the DCMR. The same options exist for operand size and alignment in this cycle as they did in the dual address source read cycle. When the complete operand is written, the DAPR is incremented by 1, 2, 4, or 16 according to the DCMR and the D_BYTE_C is decremented by the number of bytes transferred. If it is equal to zero and the transfer is completed with no errors, the DONE bit in the IDSR is set. Refer to **Section 16.6.3.2 IDMA Parameter RAM Memory Map** for more information.

Regardless of the source size, destination size, source starting address, or destination starting address, IDMA uses the most efficient packing algorithm possible to perform the transfer in the lowest number of bus cycles.

16.6.3.11.3 Single-Address Mode (Fly-By Transfers). Each IDMA channel can be independently programmed to provide single-address transfers. The internal storage is not used by IDMA, since the transfer occurs directly from a device to memory. This mode is often referred to as fly-by mode because the internal storage is not used. The external request is used to start a transfer when the single-address mode is selected. The TYPE field in the DCMR controls whether a source read or destination write cycle occurs on the data bus. If the TYPE field = 01, the external handshake signals are used with the source operand and a single-address source write occurs. If the TYPE field = 10, the external handshake signals are used with the destination operand and a single-address destination read occurs. There are two types of single-address mode cycles:

- Single-address source read—During this type of IDMA cycle, the device or memory selected by the address in the SAPR, the AT field in the SFCR, and the SIZE field in the DCMR provides the data and control signals on the data bus. This bus cycle operates like a normal read bus cycle. The destination device is controlled by the $\overline{\text{DREQx}}$ and $\overline{\text{SDACKx}}$ signals. Asserting $\overline{\text{SDACKx}}$ gives you write control to the destination device. For more details about IDMA handshake signals, refer to **Section 16.6.2 IDMA Interface Signals**. For specific timing parameters, visit our website.

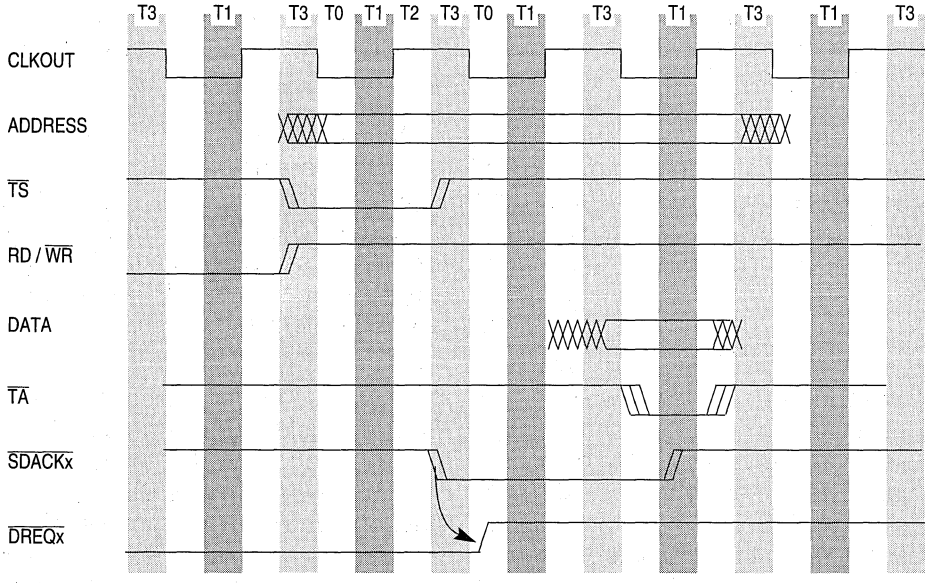


Figure 16-39. Single-Address, Peripheral Write, Asynchronous $\overline{\text{TA}}$

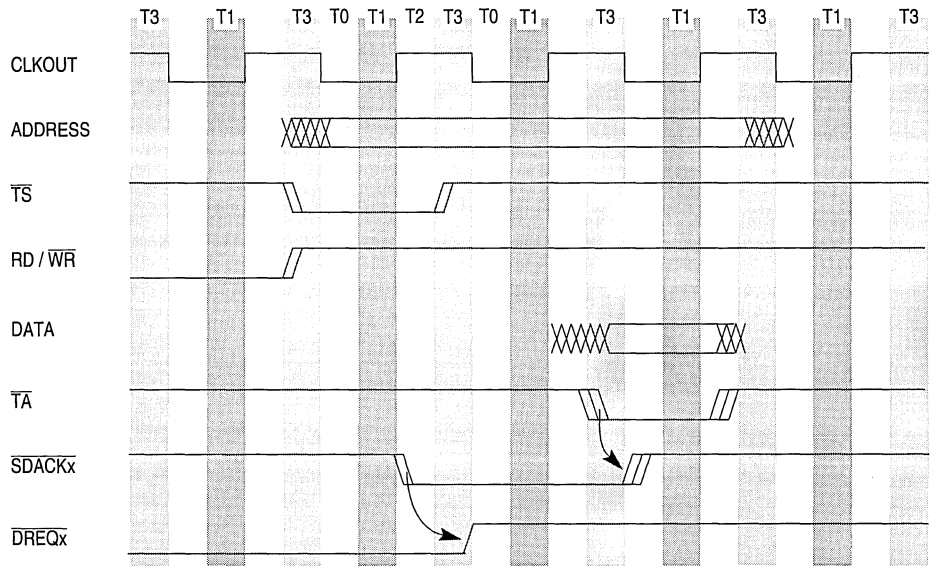


Figure 16-40. Single-Address, Peripheral Write, Synchronous TA

- Single address destination write—During this type of IDMA cycle, the source device is controlled by the IDMA handshake signals (DREQx and SDACKx). When the source device requests service from the IDMA channel, IDMA asserts SDACKx to allow the source device to drive data onto the data bus. The data is written to the device or memory selected by the AT field in the DAPR, the destination AT field in the DFCR, and the SIZE field in the DCMR. The data bus is driven to three-state for this write cycle. For more details about IDMA handshake signals, refer to **Section 16.6.2 IDMA Interface Signals**. For specific timing parameters, visit our website.

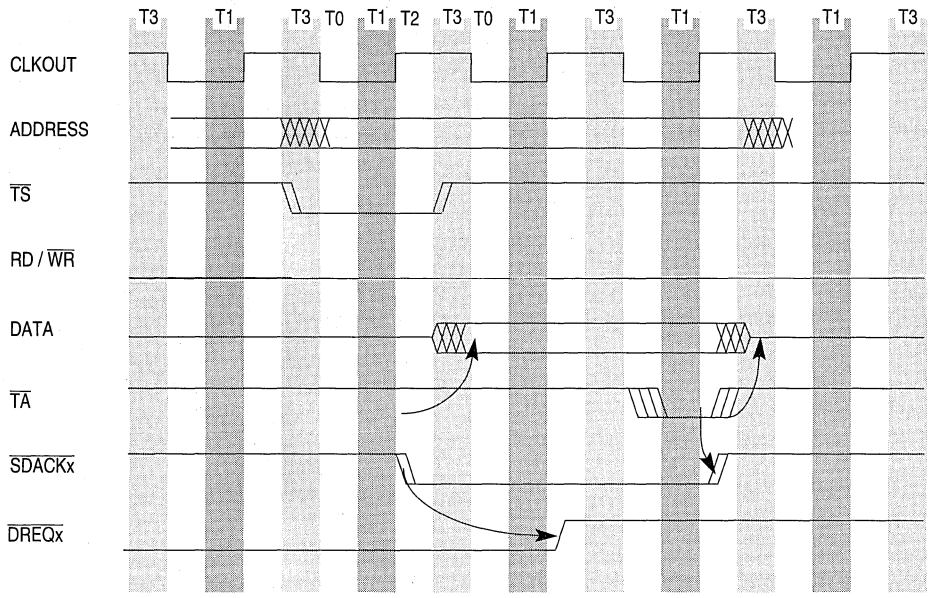


Figure 16-41. Single-Address, Peripheral Read, Synchronous TA

16.6.3.11.4 Single-Buffer Burst Fly-By Mode. This mode is used to transfer a data block from a peripheral to system memory. When the buffer has been completely transferred, channel operation is terminated. Both progressive (non-interlaced) and interlaced destination address generation modes are supported. This mode of operation is a subset of the buffer chaining mode with reduced latency and it is restricted to fly-by transfers. It is supported only on IDMA channel 1.

Progressive address generation mode is selected by setting the EIE bit in the RCCR. As a result, the organization of its parameter RAM is different from the other modes.

Interlaced address generation mode supports charge-coupled devices (CCDs), which use an interlaced readout scheme. CCDs are often used in digital cameras. The frame buffer in memory must be progressive to decrease the amount of processing the software has to perform. The interlace mode allows you to read an interlaced CCD into a progressive buffer. For example, for a 2-field CCD, the IDMA reads a line, skips a line, reads a line until the first field (the odd lines) is read. Then the process repeats for the second field (even lines). There are, however, 4-field CCDs, in which case you would read a line and then skip three. A field is a group of scan lines (odd or even) and a frame is composed of several fields. The refresh (of the display) or readout (of the CCD) is done by field sequence—field1, then field2, etc.

Interlaced mode can be set up by:

1. Initializing the IDMA single buffer mode parameter RAM.
2. In the RCCR, set the EIE bit to 1. Set the DR1M bit according to the requested mode. See **Section 16.2.6 RISC Controller Configuration Register** for more information.
3. If edge-sensitive mode is selected, program the EDM15 bit to the requested mode (falling edge/any edge).
4. In the PCSO register set the DREQ1 bit to 1. See **Section 16.14.9.4 Port C Special Options Register** for more information.

When configured to the single buffer mode, the IDMA parameter RAM is overlayed with the specific parameters in Table 16-22.

Table 16-22. Single-Buffer Mode Parameter RAM Map

ADDRESS	NAME	WIDTH	DESTINATION ADDRESS GENERATION MODE	
			PROGRESSIVE	INTERLACED
DMA Base + 00	**BAPR***	Word	Buffer Address Pointer	Current Buffer Address Pointer
DMA Base + 04	**BCR***	Word	Byte Count Register	Current Bytes Per Line
DMA Base + 08	**DCMR***	Word	DMA Channel Mode Register	DMA Channel Mode Register
DMA Base + 0C	**FBAR***	Word	Not Assigned	Field Base Address Register
DMA Base + 10	RES	Half-Word	Reserved	Reserved
DMA Base + 12	**NFLD***	Half-Word	Not Assigned	Number of Fields Register
DMA Base + 14	**LCR**	Half-Word	Not Assigned	Lines Per Field Count Register
DMA Base + 16	**L_CNT***	Half-Word	Not Assigned	Current Line Count
DMA Base + 18	**BPLR**	Half-Word	Not Assigned	Bytes Per Line Register
DMA Base + 1A	**RBR**	Half-Word	Not Assigned	Raw Bytes Register
DMA Base + 1C to DMA Base + 3F	RES	Word	Reserved	Reserved

NOTE: You are only responsible for initializing the items in bold.
* = Modified by the IDMA controller during operation and should be reinitialized before starting a new IDMA transaction.
DMA base = (IMMR & 0xFFFF0000) + 0x3CC0 (IDMA1).
All references to registers in the parameter RAM table are actually implemented in the dual-port RAM area as a memory-based register.

You must initialize the parameter RAM values before the channel is enabled. However, they should only be modified when there is no DMA activity.

- Buffer Address Pointer—The buffer address pointer contains 32 address bits of the destination buffer address used by the IDMA. The BAPR should be programmed to burst-aligned address. In progressive mode, the BAPR is incremented by 16 bytes after each transfer and it is incremented by 16 for each burst. In interlaced mode the BAPR is incremented by 16 for each burst while it is within a line and then it is incremented by the RBR to point to the next line.
- Byte Count Register—The 32-bit byte count register specifies the number of bytes to be transferred by the IDMA. The BCR is decremented by 16 bytes after each transfer and must be programmed as a multiple of 16. In progressive mode, the IDMA channel will terminate the transfer of a block of data if this register reaches zero during operation.
- DMA Channel Mode Register—The 32-bit DMA channel mode register controls the channel operation mode.

DCMR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	MB1	RESERVED		BO		AT			STR	RESERVED			EDGE	ITLC	BPR	
RESET	0	0		0		0			0	0			0	0	0	
R/W	R/W	R/W		R/W		R/W			R/W	R/W			R/W	R/W	R/W	
ADDR	(IMMR & 0xFFFF0000) + 0x3CC8															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x3CCA															

MB1—Must Be 1
For DMA operation, this bit must be set to 1.

Bits 1, 2, 9–11, 16–31—Reserved
These bits are reserved and should be set to 0.

BO—Byte Ordering

You should set this field to select the required byte ordering of the for the data buffer. If this field is modified on-the-fly, it will take effect at the beginning of the next buffer descriptor.

00 = DEC/Intel convention is used for byte ordering (swapped operation). It is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed as compared to the Motorola mode. This mode is supported only for 32-bit port size memory.
01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least significant byte of the buffer double-word contains data to be transmitted earlier than the most significant byte of the same buffer double word.
1X = Motorola byte ordering (normal operation). It is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most significant byte of the buffer word contains data to be transmitted earlier than the least significant byte of the same buffer word.

AT—Address Type 1–3

This field contains the function code value used during this SDMA channel memory access. AT0 will be driven with a one to identify this SDMA channel access as a DMA-type access.

STR—Start

This bit enables the IDMA channel. You should set it after you program BAPR and BCR. It is cleared by the channel upon completion of the transfer and the byte count in BCR is exhausted.

0 = DMA channel is disabled.
1 = DMA channel is enabled.

EDGE—Edge-Sensitive $\overline{\text{DREQ1}}$

This bit controls whether the CPM will be interlocked to the external $\overline{\text{SDACK1}}$ signal before exiting the IDMA routine. This bit must be cleared if level-sensitive $\overline{\text{DREQ1}}$ is used. If the DR1M bit of the RCCR is set to 1, this bit must be set to 0.

0 = Level-sensitive mode. The CPM will wait until $\overline{\text{SDACK1}}$ is seen externally before executing the IDMA routine.
1 = Edge-sensitive mode. The CPM may exit the IDMA routine before $\overline{\text{SDACK1}}$ is seen externally.

ITLC—Interlaced Mode

This bit controls the destination address bit generation.

0 = Progressive (non-interlaced) address generation.
1 = Interlaced address generation.

BPR—Bursts Per Request

This field determines how many bursts will be transferred per request.

00 = One burst per request.
01 = Two burst per request.
10 = Reserved.
11 = Four bursts per request.

- Field Base Address Register—This 32-bit register specifies the field destination base address. FBAR is incremented by the number of bytes per line after each field. It is used in interlaced mode only.
- Number of Fields Per Frame Register—This 16-bit register specifies the number of field per frame. It is used in interlaced mode only. NFLD is decremented after each field.
- Lines Per Field Count Register —This 16-bit register specifies the number of lines per field. It is used in interlaced mode only.
- Lines Per Field Register—This 16-bit register specifies the number of remaining lines to the end of the field. It is used in interlaced mode only, and should be initialized to the value of the LCR. L_CNT is decremented after each line.
- Bytes Per Line Register—This 16-bit register specifies the number of bytes per line. The value must be divisible by 16 for one burst per request, by 32 for two bursts per request, or by 64 for four bursts per request. It is used in interlaced mode only.
- Raw Bytes Register—This 16-bit register specifies the number of bytes to skip from the end of one line to the beginning of the next line. It is used in interlaced mode only.

16.6.3.12 IDMA STATUS REGISTER. The 8-bit IDMA status register (IDSR) is used to report events recognized by the IDMA controller. When an event is recognized, the IDMA controller sets the corresponding bit in the IDSR. This memory-mapped register can be read at any time. A bit is reset by writing a one (writing a zero has no effect).

IDSR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED						DONE	RESERVED
RESET	0						0	0
R/W	R/W						R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x910 (IDSR1), 0x918 (IDSR2)							

Bits 0–5 and 7—Reserved

These bits are reserved and should be set to 0.

DONE—IDMA Transfer Done

This bit indicates that the IDMA channel terminated a transfer. It will be set after the byte count in BCR has reached zero.

16.6.3.13 IDMA MASK REGISTER. The 8-bit read/write IDMA mask register (IDMR) has the same bit format as the IDMA status register. If a bit in this register is a one, the corresponding interrupt in the IDSR will be enabled. If the bit is zero, the corresponding interrupt in the IDSR will be masked.

IDMR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED						DONE	RESERVED
RESET	0						0	0
R/W	R/W						R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x914 (IDMR1), 0x91C (IDMR2)							

Bits 0–5 and 7—Reserved

These bits are reserved and should be set to 0.

DONE—IDMA Transfer Done

This bit indicates when the IDMA channel terminates a transfer. It is set after servicing a buffer descriptor that has the L bit set.

16.6.3.14 SINGLE-BUFFER TIMING. A typical single-address burst timing when the IDMA is in single-buffer mode, is illustrated in Figure 16-42. The peripheral device asserts the $\overline{\text{DREQ1}}$ pin and waits for $\overline{\text{SDACK1}}$ to initiate a burst transfer to or from memory. The peripheral must negate the $\overline{\text{DREQ1}}$ pin before the last beat of the transfer. Otherwise, IDMA will assume that another DMA request is pending and will start another burst cycle right after the completion of the current transfer.

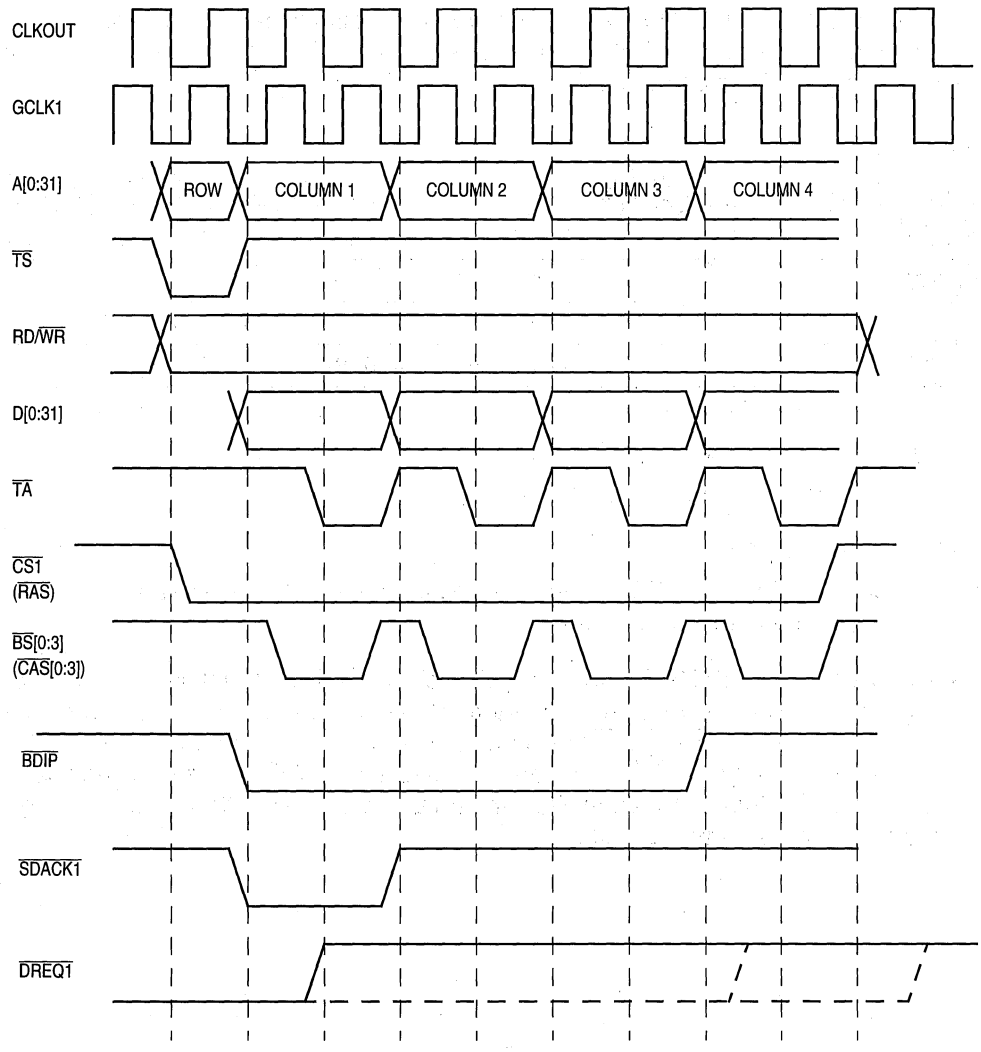


Figure 16-42. IDMA Single-Address Burst Read or Write

16.6.3.15 DOWNLOAD SEQUENCE. The microcode package is provided in S-records format. It occupies the first 512 bytes of the dual-port RAM (on the ADS address 0x2202000 to address 0x22021FF) and 256 bytes at the end of the first 4K (on the ADS address 0x2202F00 to address 0x2202FFF). For MPC823 silicon versions 0.3 (Z3, mask number 3F98S) and later, you do not need to download this microcode. Use the following MPC8bug debugger commands to load the package:

```
rms cpm rccr 0
load interlaced_dma.srx
rms cpm rccr 9
```

16.6.3.16 BUS EXCEPTIONS. When IDMA has the bus and is performing operand transfers, bus exceptions can occur. When a synchronous bus structure like those supported by the MPC823 is used, you can make provisions that allow a bus master to detect and respond to errors during a bus cycle. IDMA recognizes the same bus exceptions that the core recognizes at reset or when a transfer error occurs.

- Reset—On an external reset, IDMA immediately aborts channel operation, returns to the idle state, and clears the IDSR. If a bus cycle is in progress when reset is detected, the cycle is terminated, the control and address/data pins are three-stated, and bus ownership is released.
- Transfer error—When a fatal error occurs during a bus cycle, a bus error exception is used to abort the cycle and systematically terminate that channel operation. The IDMA terminates the current bus cycle, signals an error in the SDSR, and signals an interrupt if the corresponding bit in the SDMR is set. IDMA waits for the RISC microcontroller to reset before starting any new bus cycles. It should be noted that any data previously read from the source into the internal storage is lost.

Note: Any device that is the source or destination of the operand under IDMA handshake control for single address transfers may need to monitor $\overline{\text{TEA}}$ to detect a bus exception for the current bus cycle. $\overline{\text{TEA}}$ terminates the cycle immediately and negates SDACKx, which is used to control the transfer to or from the device.

16.7 THE SERIAL INTERFACE WITH TIME-SLOT ASSIGNER

The serial interface connects the physical layer serial lines to the serial communication controller and two serial management controllers. In its simplest configuration, the serial interface allows these controllers to be connected to their own set of individual pins. The serial communication controller or serial management controller that connects to the external world in this way connects to a nonmultiplexed serial interface (NMSI). In an NMSI configuration, the serial interface provides a flexible clocking assignment for the serial communication controller or serial management controller from a bank of external clock pins and/or internal baud rate generators.

However, the main feature of the serial interface is its time-slot assigner (TSA), which allows any combination of the serial communication or management controllers to multiplex their data together on one or one time-division multiplexed (TDM) channel. Common examples of TDMs are the T1 lines in the U.S. or Japan and the CEPT lines in Europe. Even if the time-slot assigner is not used in its intended capacity, it can still be used to generate complex waveforms on four output pins. For example, these pins can be programmed by the time-slot assigner to implement stepper motor control or variable duty cycle and period control on these pins. Any programmed configuration can be changed on-the-fly. The serial interface block diagram is illustrated in Figure 16-43.

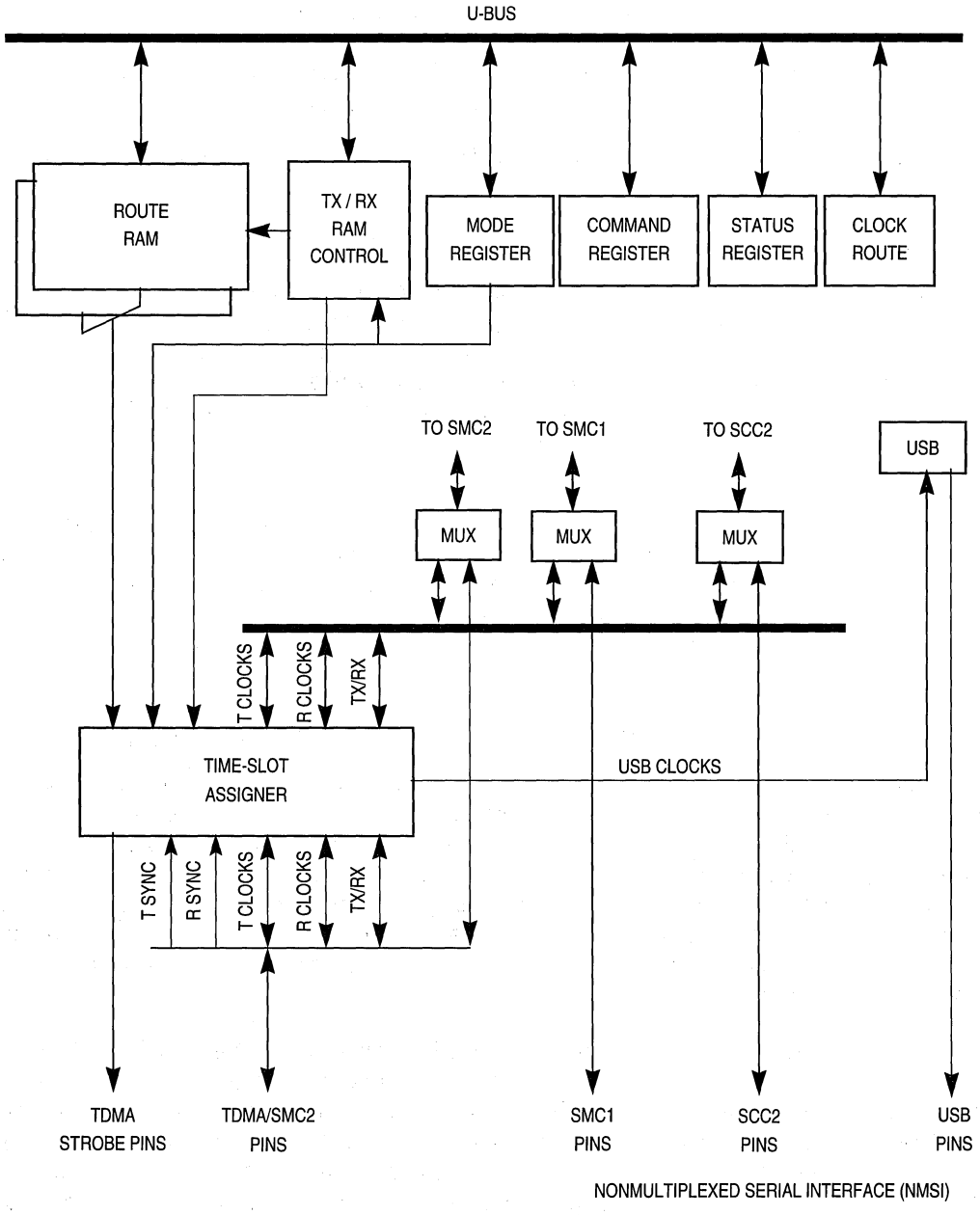


Figure 16-43. Serial Interface Block Diagram

16.7.1 Features

The top-level features of the serial interface are the time-slot assigner and NMSI. The time-slot assigner has the following main features:

- Ability to connect to an independent TDMA channel
- Independent, programmable transmit and receive routing paths
- Independent transmit and receive frame syncs
- Independent transmit and receive clocks
- Selection of rising/falling clock edges for the frame sync and data bits
- Supports 1× and 2× input clocks (1 or 2 clocks per data bit)
- Selectable delay (0–3 bits) between frame sync and frame start
- Eight programmable strobe outputs (LST[1-4] used for receive (RX) RAM while LST[5-8] is used for transmit (TX) RAM)
- 1- or 8-bit resolution in routing, masking, and strobe selection
- Supports frames up to 8,192 bits long
- Internal routing and strobe selection can be dynamically programmed
- Supports automatic echo and loopback mode for the TDMA

The NMSI has the following main features:

- The serial communication controller (SCC2) and each serial management controller (SMC) can be independently programmed to work with its own set of pins in a nonmultiplexed manner
- The serial communication controller can have its own set of modem control pins
- Each serial management controller can have its own set of pins
- The serial communication controller, universal serial bus, and each serial management controller can derive clocks externally from a bank of four clock pins or a bank of four baud rate generators

16.7.2 Configuring the Time-Slot Assigner

The time-slot assigner implements both internal route selection and time-division multiplexing for multiplexed serial channels. It supports the serial bus rate and format for most standard TDM buses, including the T1 and CEPT highways, pulse code modulation (PCM) highway, and ISDN buses in both basic and primary rates. The two popular ISDN basic rate buses—interchip digital link and general circuit interface (also known as IOM-2) are supported.

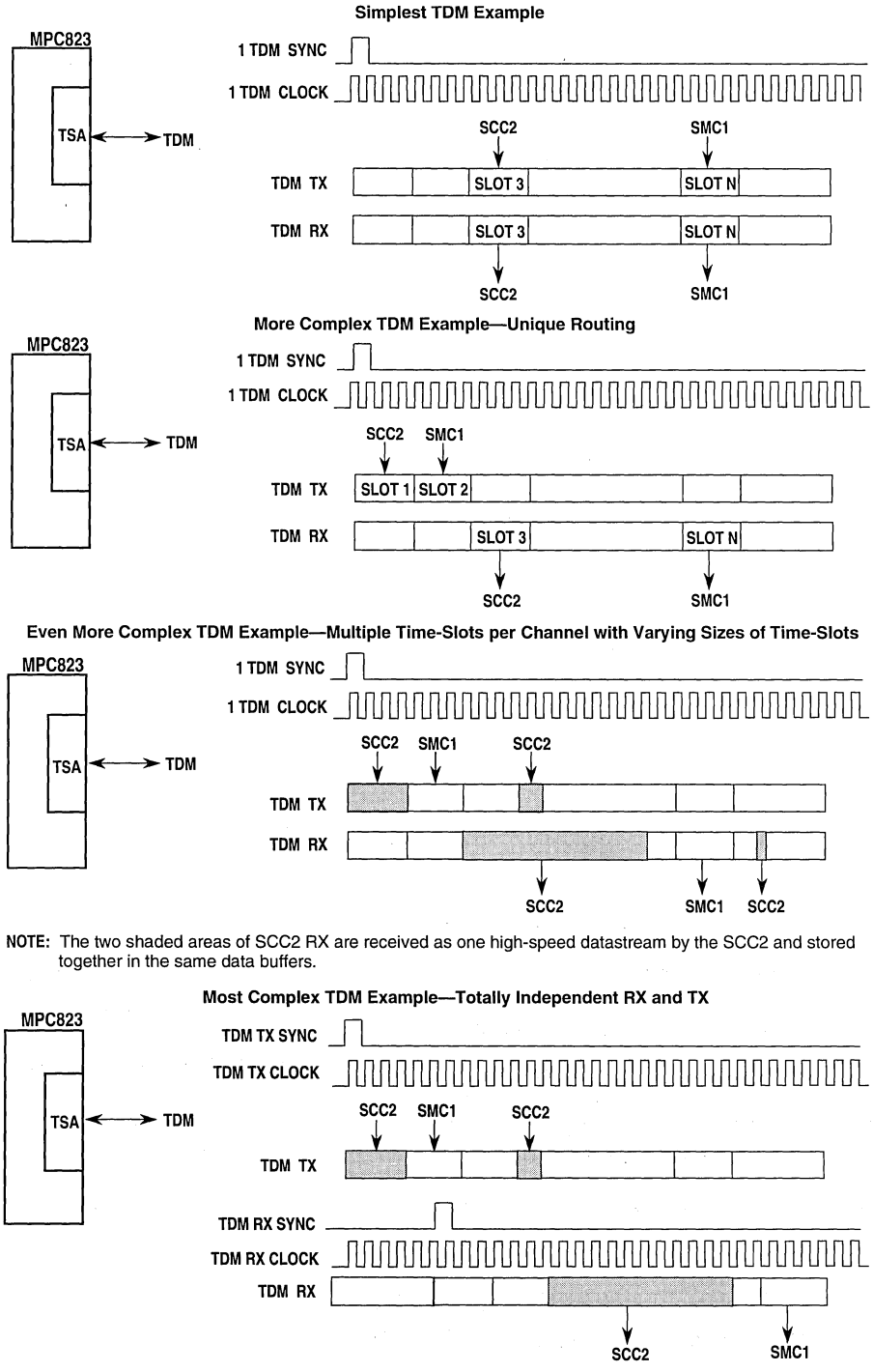
Time-slot assigner programming is completely independent of the protocol used by the serial communication controller or serial management controller. For instance, the fact that SCC2 can be programmed for the HDLC protocol has no impact on time-slot assigner programming. The purpose of the time-slot assigner is to route the data from the specified pins to the serial communication controller or serial management controller at the correct time, but it is the responsibility of the SCC2 or SMC to handle the received data. In its simplest mode, the time-slot assigner identifies the frame using one sync pulse and one clock signal that you provide. This can be enhanced to allow independent routing of the receive and transmit data on the TDMA channel. Additionally, the definition of a time-slot need not be limited to 8 bits or even to a single contiguous position within the frame. You can provide separate receive and transmit syncs as well as clocks. These various configurations are illustrated in Figure 16-44.

The time-slot assigner can support two, independent, half-duplex TDMA sources, one in reception and one in transmission, using two sync inputs and two input clocks. In addition to channel programming, the time-slot assigner supports up to eight strobe outputs that may be asserted on a bit or byte basis. These strobes are completely independent from the channel routing used by SCC2 and the SMCs. They are useful for interfacing to other devices that do not support the multiplexed interface or for enabling/disabling three-state I/O buffers in a multi-transmitter architecture.

Most time-slot assigner programming is accomplished in two 64 × 16-bit serial interface RAMs that are directly accessible by the host CPU in the internal register section of the MPC823 and are not associated with the dual-port RAM. One serial interface RAM is always used to program the transmit routing and the other is used to program the receive routing. With the serial interface RAMs, you can define the number of bits or bytes to be routed to the serial communication controller or serial management controller and decide when the external strobes are to be asserted and negated.

The size of the serial interface RAM that is available for time-slot programming depends on the configuration of the RDM field in the SIGMR. If on-the-fly changes are allowed, the serial interface RAM entries are reduced by one-half. However, the serial interface RAM size is sufficient to allow extensive time-slot programming flexibility. The maximum frame length that can be supported in any configuration is 8,192 bits. The serial interface supports two testing modes—echo and loopback. The echo mode provides a return signal from the physical interface by retransmitting the signal it has received.

The physical interface echo mode differs from the individual SCC2 echo mode in that it can operate on the entire TDM signal, rather than just on a particular SCC2 channel. The loopback mode causes the physical interface to receive the same signal it is transmitting. The serial interface loopback mode checks more than the individual SCC2 loopback does. It checks the serial interface and the internal channel routes. The maximum external serial clock that may be an input to the time-slot assigner is GCLK2 ÷ 2.5. If a serial communication controller or serial management controller is operating with the NMSI, then the serial clock rate may be slightly faster at a value not to exceed GCLK2 ÷ 2.



NOTE: The two shaded areas of SCC2 RX are received as one high-speed datastream by the SCC2 and stored together in the same data buffers.

Figure 16-44. Various Configurations With the TDM Channel

16.7.3 Enabling Connections to the Time-Slot Assigner

Each serial communication controller and serial management controller can be independently enabled to connect to the time-slot assigner. SCC2 is connected to the time-slot assigner by setting the SC2 bit in the SICR. The serial management controllers are connected to the time-slot assigner by setting the SMCx field of the SIMODE register. Additionally, the TDMA interface must be enabled before it can be connected to the time-slot assigner by setting the ENA field in the SIGMR. Once the connections are made, the exact routing decisions are made in the serial interface RAM. Refer to Figure 16-45 for more information.

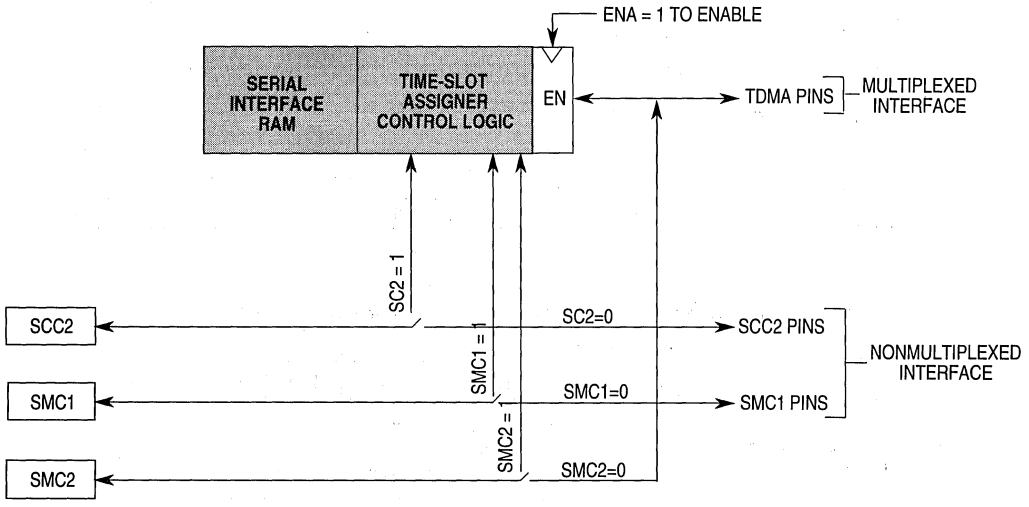


Figure 16-45. Enabling Connections Through the Serial Interface

16.7.4 Serial Interface RAM Operation

The serial interface has two 64 × 16 static RAMs that are used to control the routing of the TDMA channel to the serial communication and management controllers. These RAMs are uninitialized after power-on and, to avoid unwanted results, the host CPU should program them before enabling the multiplexed channels. The RAMs consist of 16-bit entries that define the routing control and each entry can control anywhere from 1 to 16 bits or 1 to 16 bytes. In addition to the routing, up to four strobe pins (all active high) can be asserted, depending on how the RAM is programmed.

You can configure the serial interface RAM in the following formats to support the TDMA channel:

- One multiplexed channel with static frames
- One multiplexed channel with dynamic frames

16.7.4.1 ONE MULTIPLEXED CHANNEL WITH STATIC FRAMES. In this configuration, there are 64 entries in the serial interface RAM for transmit data and strobe routing and 64 entries for receive data and strobe routing. This configuration should be chosen when the one time-division multiplex's routing does not need to be dynamically changed.

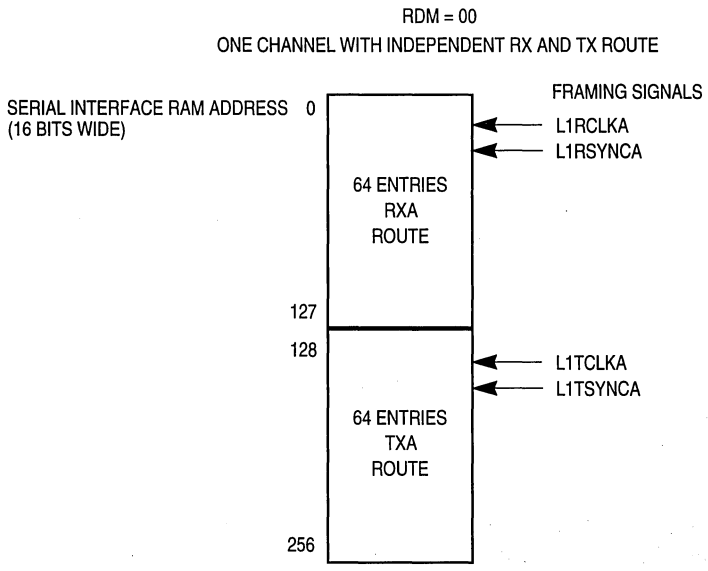


Figure 16-46. Configuring the TDM with Static Frames

16.7.4.2 ONE MULTIPLEXED CHANNEL WITH DYNAMIC FRAMES. In this configuration, there is one multiplexed channel and it has 32 entries for transmit data and strobe routing and 32 entries for receive data and strobe routing. In each RAM, one of the partitions is the current-route RAM and the other is a shadow RAM that allows you to change the serial routing. After programming the shadow RAM, set the CSRRA bit of the associated channel in the SIMCR to receive and the CSRTA bit to transmit. When the next frame sync arrives, the serial interface automatically exchanges the current-route RAM for the shadow RAM. Refer to **Section 16.7.4.4 Serial Interface RAM Dynamic Changes** for more details on how to dynamically change the channel route. You should only use this configuration when the routing on the time-division multiplex needs to be dynamically changed.

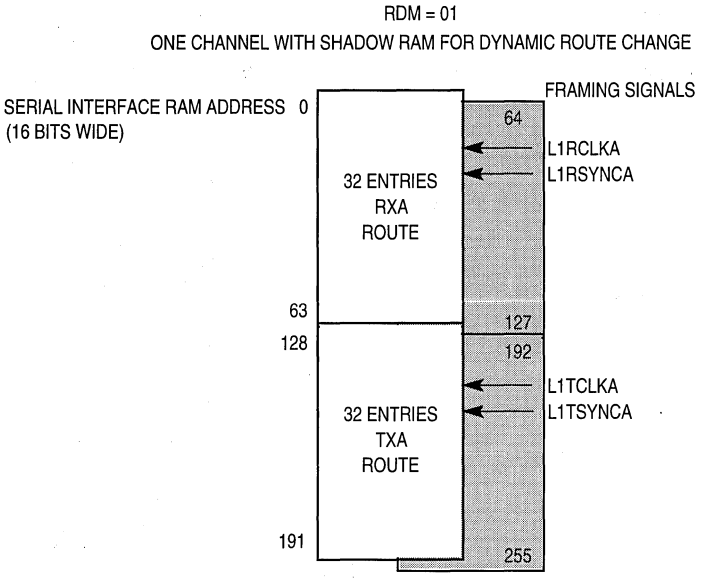


Figure 16-47. Configuring the TDM with Dynamic Frames

16.7.4.3 PROGRAMMING THE SERIAL INTERFACE RAM ENTRIES. The programming of each word within the RAM determines the routing of the serial bits and assertion of strobe outputs. The RAM programming codes are shown in the following table.

SERIAL INTERFACE RAM ENTRIES

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	LOOP	SWTR	SSEL1	SSEL2	SSEL3	SSEL4	RES	CSEL			CNT				BYT	LST
RESET	0	0	0	0	0	0	0	0			0				0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W			R/W				R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xC00 TO 0xDFF															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED															
RESET	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0xC00 TO 0xDFF															

LOOP—Loopback on This Time-Slot

0 = Normal mode.
1 = Loopback mode for this time-slot.

SWTR—Switch Transmit and Receive

This bit is only valid in the receive route RAM and is ignored in the transmit route RAM. This bit affects the operation of both the L1RXDA and L1TXDA pins. The SWTR bit is only set in special situations where you prefer to receive data from a transmit pin and transmit data on a receive pin. For instance, consider the situation where devices A and B are connected to the same time-division multiplex, each with different time-slots. Normally, there is no opportunity for stations A and B to communicate with each other directly over the time-division multiplex since they both receive the same TDM receive data and transmit on the same TDM transmit signal.

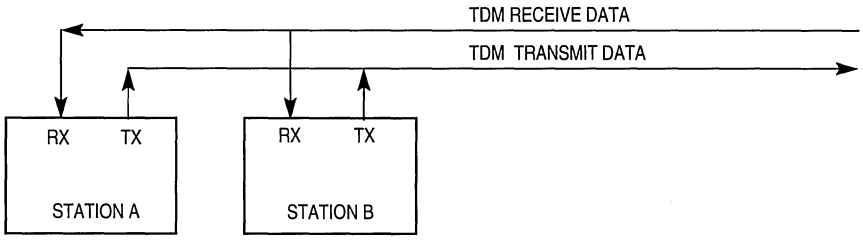


Figure 16-48. Using the SWTR Bit

The SWTR option gives station B the opportunity to listen to transmissions from station A and transmit data to station A. To do this, station B would set the SWTR bit in its receive route RAM. For this entry, receive data is taken from the L1TXDA pin and data is transmitted on the L1RXDA pin. If you only want to listen to station A transmissions and not transmit data on L1RXDA, then you should clear the CSEL field in the corresponding transmit route RAM entry to prevent transmission on the L1RXDA pin.

It is also possible for station B to transmit data to station A by setting the SWTR bit of the entry in its receive route RAM. Data is transmitted on the L1RXDA pin rather than the L1TXDA pin, according to the transmit route RAM. This configuration could, however, cause collisions with other data on the L1RXDA pin unless care is taken to choose an available (quiet) time-slot. If you only want to transmit on L1RXDA and not receive data on L1TXDA, then you should clear the CSEL field in the receive route RAM to prevent reception of data on L1TXDA.

0 = Normal operation of the L1TXDA and L1RXDA pins.
1 = Data is transmitted on the L1RXDA pin and is received from the L1TXDA pin for the duration of this entry.

Note: If the transmit and receive sections of the TDMA do not use a single clock source, the SWTR feature can cause erratic results to occur.

SSEL1–SSEL4—Strobe Select 1–4

The four strobes—L1ST1, L1ST2, L1ST3, and L1ST4—can be assigned to the receive RAM and asserted or negated with L1RCLKA. L1ST5, L1ST6, L1ST7, and L1ST8 can be assigned to the transmit RAM and asserted or negated with L1TCLKA. Each bit corresponds to the value the strobe should have during this bit/byte group. Multiple strobes can be asserted simultaneously, if preferred. If a strobe is configured to be asserted in two consecutive serial interface RAM entries, then it remains continuously asserted while the serial interface RAM entries are being processed. If a strobe is asserted on the last entry in the table, the strobe is negated after the last entry finishes processing.

Note: Each strobe is changed with the corresponding RAM clock and is only output if the corresponding parallel I/O is configured as a dedicated pin.

Bit 6—Reserved

This bit is reserved and should be set to 0.

CSEL—Channel Select

000 = The bit/byte group is not supported by the MPC823. The transmit data pin is three-stated and the receive data pin is ignored.
001 = Reserved.
010 = The bit/byte group is routed to SCC2.
011 = Reserved.
100 = Reserved.
101 = The bit/byte group is routed to SMC1.
110 = The bit/byte group is routed to SMC2.
111 = The bit/byte group is not supported by the MPC823. This code is also used in the SCIT mode as the D channel grant.

CNT—Count

This value indicates the number of bits/bytes (according to the BYT bit) that the routing and strobe select of this entry controls. If CNT = 0000, then 1 bit/byte is chosen and if CNT = 1111 16 bits/bytes are selected.

BYT—Byte Resolution

0 = Bit resolution. The value of CNT indicates the number of bits in this group.
1 = Byte resolution. The value of CNT indicates the number of bytes in this group.

LST—Last Entry in the RAM

Whenever the serial interface RAM is used, this bit must be set in one of the TX or RX entries of each group. Even if all entries of a group are used, this bit must still be set in the last entry.

0 = This is not the last entry in this section of the route RAM.
1 = This is the last entry in this RAM. After this entry, the serial interface waits for the sync signal to start the next frame.

Note: If a second sync signal is received before the end of a frame (as defined by the last serial interface RAM entry), an error occurs. The serial interface will terminate RAM processing and cease transmitting or receiving data until a third sync signal is received.

16.7.4.4 SERIAL INTERFACE RAM DYNAMIC CHANGES. The serial interface RAM has two operating modes:

- A time-division mulitplex channel with a static routing definition. The serial interface RAM is divided into two parts (RX and TX).
- A time-division mulitplex channel that allows dynamic changes. The serial interface RAM is divided into four parts.

Dynamic changes allow the routing definition of a TDMA to be modified while the serial communication controller and serial management controllers are connected to it. With fixed routing, a change has three requirements that must be met before the new routing takes effect:

- The serial communication controller and serial management controllers connected to the time-slot assigner must be disabled.
- The serial interface routing must be modified.
- The serial communication controller and serial management controllers connected to the time-slot assigner must be reenabled.

Dynamic changes divide portions of the serial interface RAM into current-route and shadow RAM. Once the current-route RAM is programmed, the time-slot assigner and serial interface channels are enabled and time-slot assigner operation begins. When you need to make a change in routing, you must program the shadow RAM with the new route and set the CSRRA bit in the SIMCR to receive and the CSRTA bit to transmit. As a result, the serial interface exchanges the shadow RAM and the current-route RAM as soon as the corresponding sync arrives and resets the appropriate CSRxA bit to signify that the operation has completed. At this time, you can change the routing again. Notice that the original current-route RAM is now the shadow RAM and vice versa. Figure 16-49 illustrates an example of the shadow RAM exchange process.

If a TDMA with dynamic changes is programmed, the initial current-route RAM addresses in the serial interface RAM are as follows:

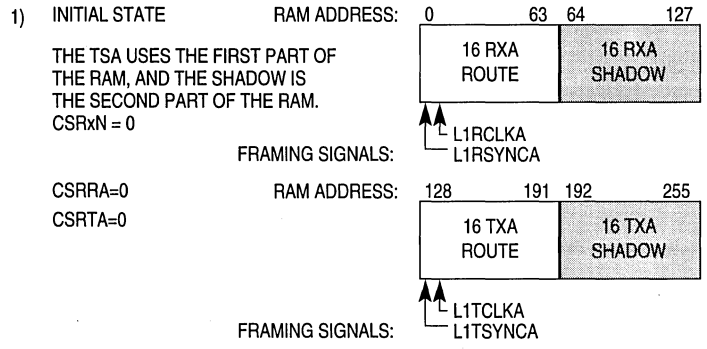
- 0–63 RXa Route
- 128–191 TXa Route

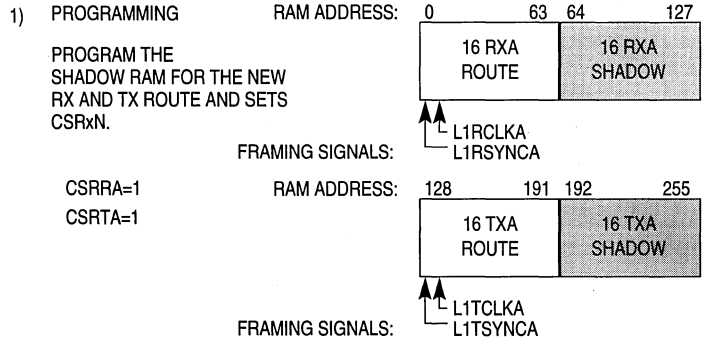
The shadow RAMs are at addresses:

- 64–127 RXa Route
- 192–255 TXa Route

You can read any RAM at any time, but for proper serial interface operation you must not try to write the current-route RAM. You can, however, read the serial interface status register (SISTR) to find out which part of the RAM is the current-route RAM.

Besides knowing which RAM is the current-route RAM, you might need to know which entry the time-slot assigner is currently using within the current-route RAM. This information is provided in the SIRP register, which is described in detail in **Section 16.7.5.6 Serial Interface RAM Pointer Register**. You can also externally connect one of the eight strobes to an interrupt pin to generate an interrupt on a particular serial interface RAM entry starting or ending execution by the time-slot assigner.





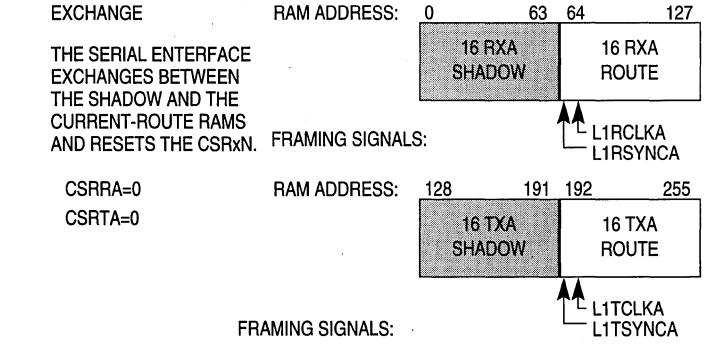


Figure 16-49. Serial Interface RAM Dynamic Changes

16.7.5 Serial Interface Programming Model

16.7.5.1 SERIAL INTERFACE GLOBAL MODE REGISTER. The 8-bit, memory-mapped, read/write serial interface global mode register (SIGMR) defines the RAM division modes.

SIGMR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED					ENA	RDM	
RESET	0					0	0	
R/W	R/W					R/W	R/W	
ADDR	(IMMR & 0xFFFF0000) + 0xAE4							

Bits 0–4—Reserved

These bits are reserved and should be set to 0.

ENA—Enable Channel A

0 = Channel A is disabled. The serial interface RAMs and TDMA routing are in a state of reset, but all other serial interface functions still operate.
1 = The serial interface is enabled.

RDM—RAM Division Mode

This field defines the RAM division mode and the number of multiplexed channels supported in the serial interface.

00 = The serial interface supports one TDMA channel with 64 entries for receive routing and another 64 for transmit routing.
01 = The serial interface supports one TDMA channel with 32 entries for receive routing and another 32 for transmit routing. There are an additional 32 shadow entries for the receive routing and 32 more for transmit routing that can be used to dynamically change the routing.
1x = Reserved.

16.7.5.2 SERIAL INTERFACE MODE REGISTER. The 32-bit, memory-mapped, read/write serial interface mode register (SIMODE) defines the serial interface operation modes and with serial interface RAM allows you to support any or all of the ISDN channels independently when in IDL or GCI mode.

SIMODE

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	SMC2	SMC2CS			RESERVED											
RESET	0	0			0											
R/W	R/W	R/W			R/W											
ADDR	(IMMR & 0xFFFF0000) + 0xAE0															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	SMC1	SMC1CS			SDMA		RFSDA		DSCA	CRTA	STZA	CEA	FEA	GMA	TFSDA	
RESET	0	0			0		0		0	0	0	0	0	0	0	
R/W	R/W	R/W			R/W		R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W	
ADDR	(IMMR & 0xFFFF0000) + 0xAE2															

SMC2—SMC2 Connection

0 = NMSI mode. The clock source is determined by the SMC2CS field and the data comes from a dedicated SMTXD2 and SMRXD2 pin in NMSI mode.
1 = SMC2 is connected to the multiplexed serial interface (TDMA channel).

SMC2CS—SMC2 Clock Source (NMSI mode)

SMC2 can take its clocks from one of the baud rate generators or one of four pins from the bank of clocks. However, the SMC2 transmit and receive clocks must be the same when they are connected to NMSI mode.

000 = SMC2 transmit and receive clocks are BRG1.
001 = SMC2 transmit and receive clocks are BRG2.
010 = SMC2 transmit and receive clocks are BRG3.
011 = SMC2 transmit and receive clocks are BRG4.
100 = SMC2 transmit and receive clocks are CLK1.
101 = SMC2 transmit and receive clocks are CLK2.
110 = SMC2 transmit and receive clocks are CLK3.
111 = SMC2 transmit and receive clocks are CLK4.

SMC1—SMC1 Connection

0 = NMSI mode. The clock source is determined by the SMC1CS field and the data comes from a dedicated pin SMTXD1 and SMRXD1 in NMSI mode.
1 = SMC1 is connected to the multiplexed serial interface (TDMA channel).

SMC1CS—SMC1 Clock Source (NMSI mode)

SMC1 can take its clocks from one of the baud rate generators or one of four pins from the bank of clocks. The SMC1 transmit and receive clocks must be the same when it is connected to the NMSI.

000 = SMC1 transmit and receive clocks are BRG1.
001 = SMC1 transmit and receive clocks are BRG2.
010 = SMC1 transmit and receive clocks are BRG3.
011 = SMC1 transmit and receive clocks are BRG4.
100 = SMC1 transmit and receive clocks are CLK1.
101 = SMC1 transmit and receive clocks are CLK2.
110 = SMC1 transmit and receive clocks are CLK3.
111 = SMC1 transmit and receive clocks are CLK4.

SDMA—Serial Interface Diagnostic Mode for TDMA

00 = Normal operation.
01 = Automatic echo. In this mode, the channel transmitter automatically retransmits the data received from the TDMA on a bit-by-bit basis. The receive section operates normally, but the transmit section can only retransmit received data. In this mode, the L1GRA signal is ignored.
10 = Internal loopback. In this mode, the TDMA transmitter output is internally connected to the TDMA receiver input (L1TXDA is connected to L1RXDA). The receiver and transmitter operate normally. The data appears on the L1TXDA pin. In this mode the L1RQA signal is asserted normally. The L1GRA signal is ignored.
11 = Loopback control. In this mode, the TDMA transmitter output is internally connected to the TDMA receiver input (L1TXDA is connected to L1RXDA). The transmitter output (L1TXDA) and the L1RQA pin is inactive. This mode is used to accomplish loopback testing of the entire TDMA without affecting the external serial lines.

Note: In modes 01,10, and 11, the receive and transmit clocks should be identical.

RFSDA—Receive Frame Sync Delay for TDMA

This field determines the number of clock delays between the receive sync and the first bit of the receive frame. Even if the CRTA bit is set, these bits do not control the delay for the transmit frame.

00 = No bit delay. The first bit of the frame is transmitted/received on the same clock as the sync. Use for GCI.
01 = 1-bit delay. Use for IDL.
10 = 2-bit delay.
11 = 3-bit delay.

See the examples in Figure 16-50 and Figure 16-51 to find out how to use these bits.

DSCA—Double Speed Clock for TDMA

This bit controls how some time-division multiplex channels, such as GCI, define the input clock to be two times faster than the data rate.

0 = The channel clock (L1RCLKA and/or L1TCLKA) is equal to the data clock. Use for IDL and most TDM formats.
1 = The channel clock rate is twice the data rate. Use for GCI.

CRTA—Common Receive and Transmit Pins for TDMA

This bit is useful when the transmit and receive sections of a given TDMA use the same clock and sync signals. In this mode, the L1TCLKA and L1TSYNCA pins can be used as general-purpose I/O pins.

0 = Separate pins. The receive section of this TDMA uses L1RCLKA and L1RSYNCA pins for framing and the transmit section uses L1TCLKA and L1TSYNCA for framing.
1 = Common pins. The receive and transmit sections of this TDMA use L1RCLKA as clock pin of the channel and L1RSYNCA as the receive and transmit sync pin. Use for IDL and GCI.

STZA—Set L1TXDA to Zero for TDMA

0 = Normal operation.
1 = L1TXDA is set to zero until serial clocks are available, which is useful for GCI activation.

CEA—Clock Edge for TDMA

0 = Data is transmitted on the rising edge of the clock and received on the falling edge (use for IDL and GCI).
1 = Data is transmitted on the falling edge of the clock and received on the rising edge.

FEA—Frame Sync Edge for TDMA

This bit indicates when the L1RSYNCA and L1TSYNCA pulses are sampled with the falling or rising edge of the channel clock.

0 = Falling edge. Use for IDL and GCI.
1 = Rising edge.

GMA—Grant Mode for TDMA

0 = GCI/SCIT mode. The GCI/SCIT D channel grant mechanism for transmission is internally supported. The grant is one bit from the receive channel. This bit is marked by programming the channel select bits of the serial interface RAM with 111 to assert an internal strobe on it. See **Section 16.7.7.2.2 SCIT Mode**.
1 = IDL mode. A grant mechanism is supported if the corresponding GR2 bit in the SICR register is set. The grant is a sample of the L1GRA pin while L1TSYNCA is asserted. This grant mechanism implies the IDL access controls for transmission on the D channel. Refer to **Section 16.7.6.2 Programming the IDL Interface** for more information.

TFSDA—Transmit Frame Sync Delay for TDMA

This field determines the number of clock delays between the transmit sync and the first bit of the transmit frame.

00 = No bit delay. The first bit of the frame is transmitted/received on the same clock as the sync.
01 = 1-bit delay.
10 = 2-bit delay.
11 = 3-bit delay.

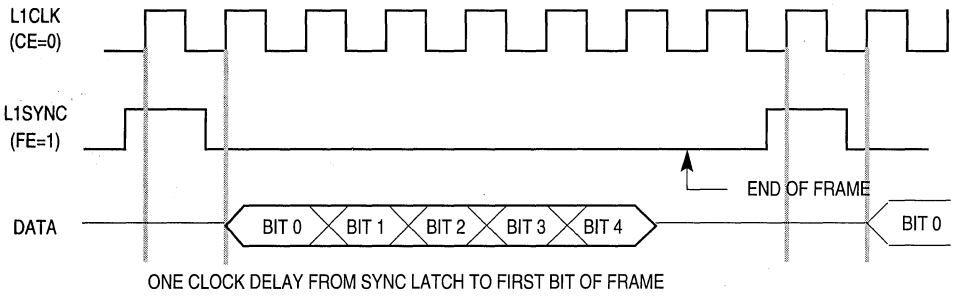


Figure 16-50. Example of One Clock Delay from Sync to Data (RFSD = 01)

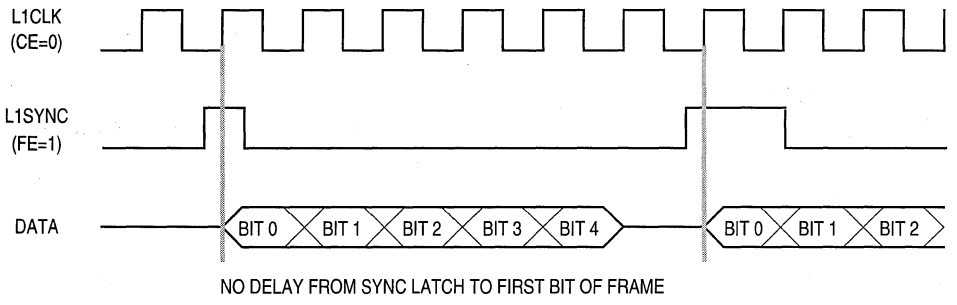


Figure 16-51. Example of No Delay from Sync to Data (RFSD = 00)

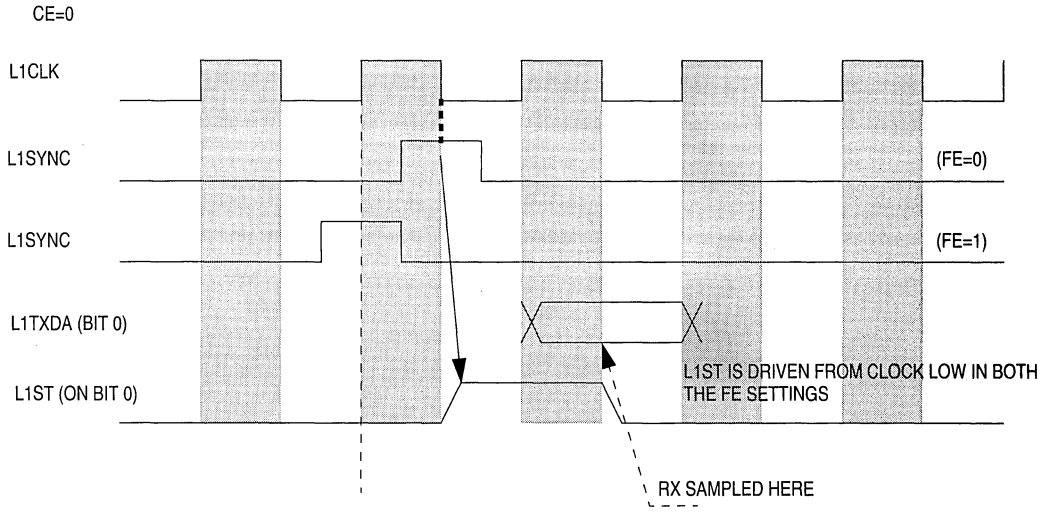


Figure 16-52. Example of Clock Edge (CE) Effect When DSC = 0

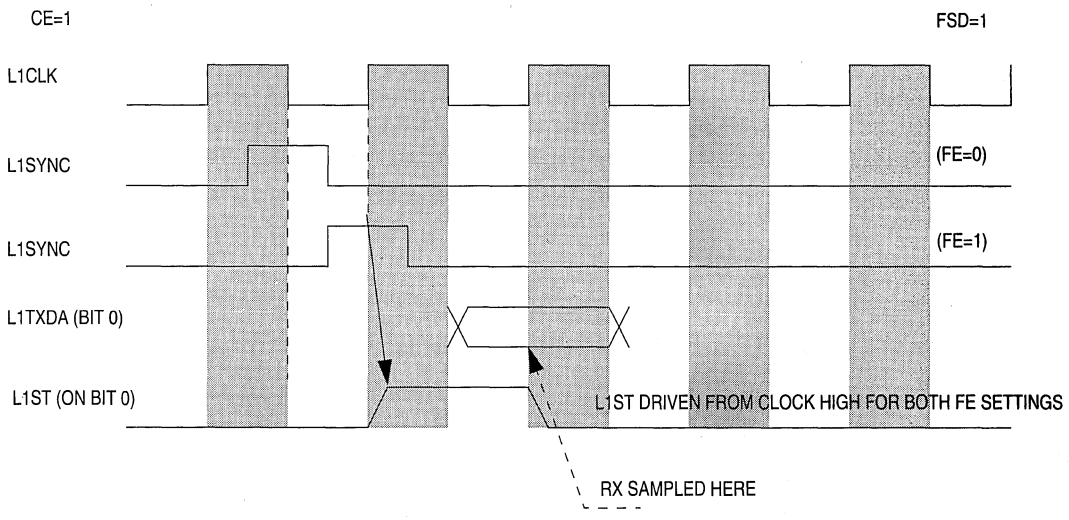


Figure 16-53. Example of Clock Edge (CE) Effect When DSC = 1

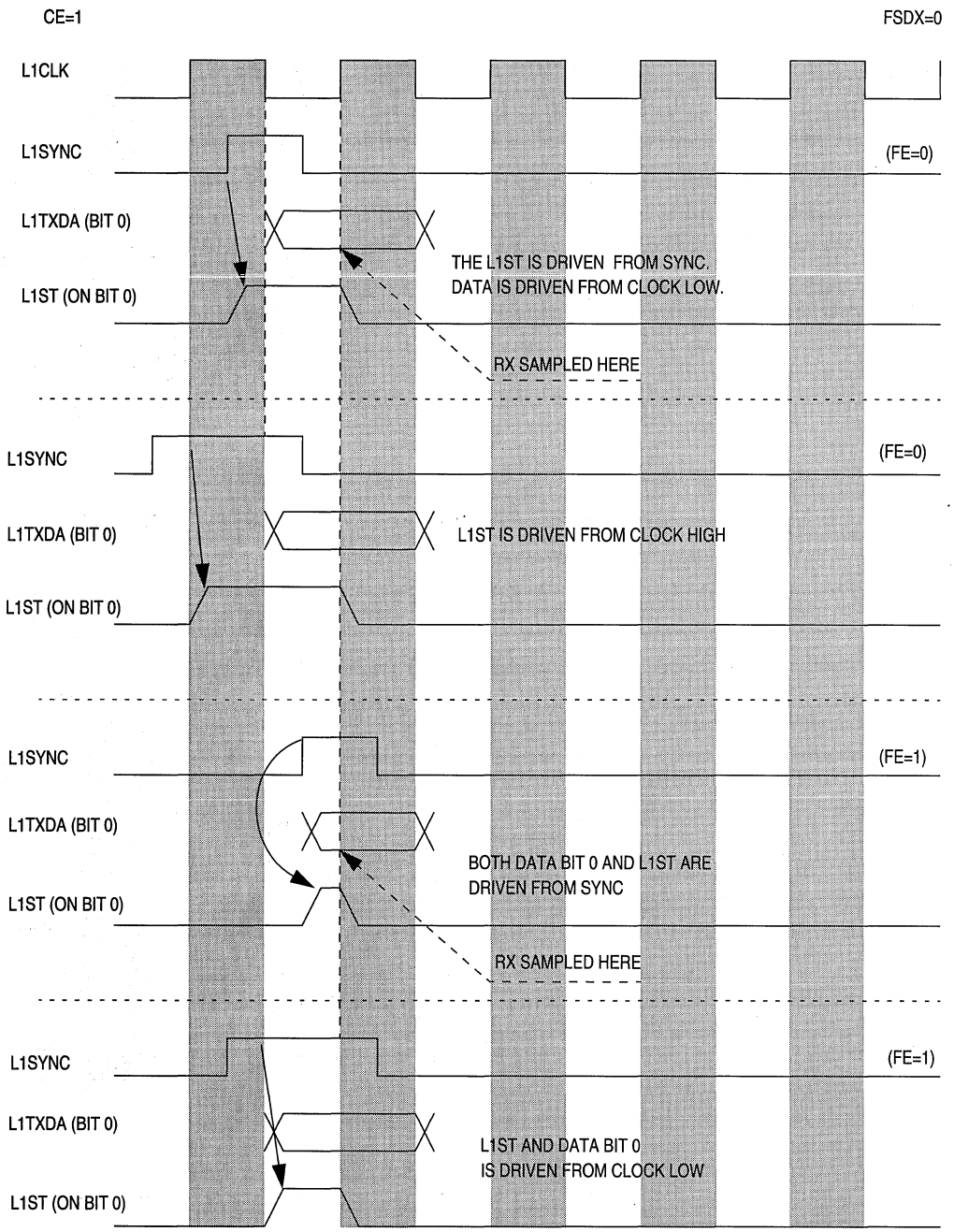


Figure 16-54. Example of Frame Transmission Reception When RFSDx or TFSDx = 0 and CD = 1

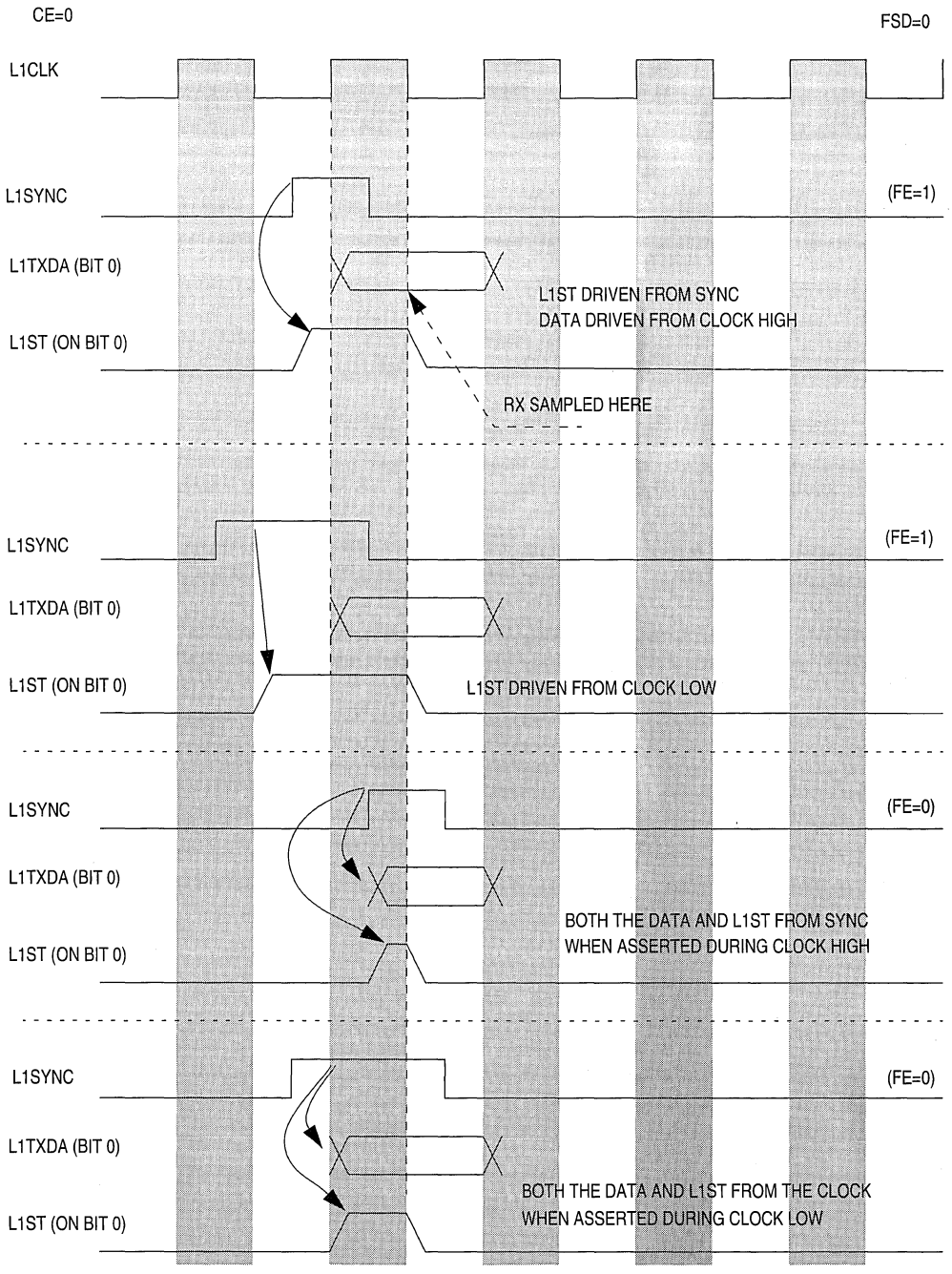


Figure 16-55. Example of CEx = 0 and FEx Interaction, XFSD = 0

16.7.5.3 SERIAL INTERFACE CLOCK ROUTE REGISTER. The 32-bit, read/write, memory-mapped serial interface clock route (SICR) register is used to define the universal serial bus and serial communication controller clock sources that can be one of the four baud rate generators or an input from a bank of clock pins.

SICR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0xAEC															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	GR2	SC2	R2CS			T2CS			RES		R1CS			RESERVED		
RESET	0	0	0			0			0		0			0		
R/W	R/W	R/W	R/W			R/W			R/W		R/W			R/W		
ADDR	(IMMR & 0xFFFF0000) + 0xAEE															

Bits 0–15—Reserved

These bits are reserved and should be set to 0.

GR2—Grant Support of SCC2

0 = SCC2 transmitter does not support the grant mechanism. The grant is always asserted internally.

1 = SCC2 transmitter supports the grant mechanism as determined by the GMA bit of the SIMODE register.

SC2—SCC2 Connection

0 = SCC2 is not connected to the multiplexed serial interface but is either connected directly to the NMSI2 pins or is not used. You can choose either the general-purpose I/O port pins or dedicated SCC2 pins in the parallel I/O port registers. See **Section 16.14 The Parallel I/O Ports** for more information.

1 = SCC2 is connected to the multiplexed serial interface. The NMSI2 receive pins are available for other purposes.

R2CS—Receive Clock Source for SCC2

This field is ignored when the SCC2 is connected to the time-slot assigner (SC2 = 1).

000 = SCC2 receive clock is BRG1.
001 = SCC2 receive clock is BRG2.
010 = SCC2 receive clock is BRG3.
011 = SCC2 receive clock is BRG4.
100 = SCC2 receive clock is CLK1.
101 = SCC2 receive clock is CLK2 .
110 = SCC2 receive clock is CLK3 .
111 = SCC2 receive clock is CLK4 .

T2CS—Transmit Clock Source for SCC2

This field is ignored when SCC2 is connected to the time-slot assigner (SC2 = 1).

000 = SCC2 transmit clock is BRG1.
001 = SCC2 transmit clock is BRG2.
010 = SCC2 transmit clock is BRG3.
011 = SCC2 transmit clock is BRG4.
100 = SCC2 transmit clock is CLK1.
101 = SCC2 transmit clock is CLK2.
110 = SCC2 transmit clock is CLK3.
111 = SCC2 transmit clock is CLK4.

Bits 24–25—Reserved

These bits are reserved and should be set to 0.

R1CS—Clock Source for the USB

000 = USB clock is BRG1.
001 = USB clock is BRG2.
010 = USB clock is BRG3.
011 = USB clock is BRG4.
100 = USB clock is CLK1.
101 = USB clock is CLK2.
110 = USB clock is CLK3.
111 = USB clock is CLK4.

Bits 29–31—Reserved

These bits are reserved and should be set to 0.

16.7.5.4 SERIAL INTERFACE COMMAND REGISTER. The 8-bit serial interface command register (SICMR) allows you to dynamically program the serial interface RAM. The contents of this register are only valid in the RAM division mode. For more information about dynamic programming, refer to **Section 16.7.4.4 Serial Interface RAM Dynamic Changes**.

SICMR

BIT	0	1	2	3	4	5	6	7
FIELD	CSRRA	CSRTA	RESERVED					
RESET	0	0	0					
R/W	R/W	R/W	R/W					
ADDR	(IMMR & 0xFFFF0000) + 0xAE7							

CSRRA—Change Shadow RAM for TDMA Receiver

When set, this bit causes the serial interface receiver to replace the current route with the shadow RAM. You set this bit and the serial interface clears it.

0 = The receiver shadow RAM is not valid. You can write into the shadow RAM to program a new routing.
1 = The receiver shadow RAM is valid. The serial interface exchanges between the RAMs and take the new receive routing from the receiver shadow RAM. This bit is cleared as soon as the switch has completed.

CSRTA—Change Shadow RAM for TDMA Transmitter

When set, this bit causes the serial interface transmitter to replace the current route with the shadow RAM. You set this bit and the serial interface clears it.

0 = The transmitter shadow RAM is not valid. You can write into the shadow RAM to program a new routing.
1 = The transmitter shadow RAM is valid. The serial interface exchanges between the RAMs and take the new transmitter routing from the receiver shadow RAM. This bit is cleared as soon as the switch has completed.

Bits 2–7—Reserved

These bits are reserved and should be set to 0.

16.7.5.5 SERIAL INTERFACE STATUS REGISTER. The 8-bit serial interface status register (SISTR) lets you know which part of the serial interface RAM is the current-route RAM. The value of this register is only valid when the corresponding bit in the SICMR is cleared.

SISTR

BIT	0	1	2	3	4	5	6	7
FIELD	CRORA	CROTA	RESERVED					
RESET	0	0	0					
R/W	R	R	R					
ADDR	(IMMR & 0xFFFF0000) + 0xAE6							

CRORA—Current Route of TDMA Receiver

0 = The current-route receiver RAM is in address 0–63 when the serial interface supports the TDM (RDM field in the SIGMR = 01).
1 = The current-route receiver RAM is in address 64–127 when the serial interface supports the TDM (RDM = 01).

CROTA—Current Route of TDMA Transmitter

0 = The current-route transmitter RAM is in address 128–191 when the serial interface supports the TDM (RDM field in the SIGMR = 01).
1 = The current-route transmitter RAM is in address 192–255 when the serial interface supports the TDM (RDM = 01).

Bits 2–7—Reserved

These bits are reserved and should be set to 0.

16.7.5.6 SERIAL INTERFACE RAM POINTER REGISTER. The 32-bit, read-only serial interface RAM pointer (SIRP) register lets you know which RAM entry is currently being serviced. It gives you a real-time status location of the serial interface that is currently inside the TDM frame. Although not everyone needs to access the SIRP register, it does provide information that might be helpful for debugging and for synchronizing system activity with TDM activity. Usually, reading the SISTR is sufficient for most applications.

You can determine which RAM entry in the serial interface RAM is currently in progress, but you cannot determine the status within that entry. If the RAM entry is programmed to select four contiguous time-slots from the TDM and the SIRP register indicates the entry is currently active, you will not know which of the four time-slots is currently in progress. The SIRP register does, however, change its status as soon as the next serial interface RAM entry begins processing.

Note: You can externally connect one of the eight strobes to an interrupt pin to generate an interrupt on a particular serial interface RAM entry.

The value of this register changes on serial clock transitions. Before acting on the information found in this register, you should perform two reads and verify that they returned the same value. The pointers provided by this register indicate the serial interface RAM entry word offset that is currently in progress.

SIRP

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED		VT2	TAPTR2					RESERVED		VT1	TAPTR1				
RESET	0		0	0					0		0	0				
R/W	R		R	R					R		R	R				
ADDR	(IMMR & 0xFFFF0000) + 0xAF0															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED		VR2	RAPTR2					RESERVED		VR1	RAPTR1				
RESET	0		0	0					0		0	0				
R/W	R		R	R					R		R	R				
ADDR	(IMMR & 0xFFFF0000) + 0xAF2															

Bits 0–1 and 8–9—Reserved

These bits are reserved and should be set to 0.

VT1, VT2, VR1, and VR2—Time-Slot Valid Bits for Serial Interface RAM Entries

The VTx or VRx bit in each entry shows that the entry is valid, which is helpful when the corresponding pointer entry value is zero. Additionally, the VTx or VRx bit saves you from having to read both the SIRP and the SISTR registers to obtain the information you need. The pointer values are based on the value of the RDM field in the SIGMR.

Bits 16–17 and 24–25—Reserved

These bits are reserved and should be set to 0.

TAPTR2, TAPTR1, RAPTR2, and RAPTR1—Serial Interface RAM Time-Slot Pointers

In all cases, the value of the TAPTRx or RAPTRx fields increment by one for each entry that the serial interface processes. Since each TAPTRx and RAPTRx is 5 bits each, the values in each field can range from 0 to 31, corresponding to 32 different serial interface RAM entries. The full pointer range may not necessarily be used. For instance, if the last bit is set in the fifth serial interface RAM entry, then the pointer only reflects values from 0 to 4, but once the fifth entry is processed by the serial interface, the pointer is reset to 0.

16.7.5.6.1 SIRP Indication When RDM = 00. 64 entries cannot be signified with a single 5-bit pointer, two 5-bit pointers are used instead—one for the first 32 entries and one for the second 32 entries. If the corresponding VTx or VRx bit is set, then:

- RAPTR1 and RAPTR2 contain the address of the currently active RX RAM entry. When the serial interface services entries 1–32, RAPTR1 is incremented and RAPTR2 is continuously cleared. When the serial interface services entries 33–64, RAPTR1 is continuously cleared and RAPTR2 is incremented.
- TAPTR1 and TAPTR2 contain the address of the currently active TX RAM entry. When the serial interface services entries 1–32, TAPTR1 is incremented and TAPTR2 is continuously cleared. When the serial interface services entries 33–64, TAPTR1 is continuously cleared and TAPTR2 is incremented.

16.7.5.6.2 SIRP Indication When RDM = 01. For the receiver, either RAPTR1 or RAPTR2 is used, depending on the portion of the serial interface RX RAM that is currently active. For the transmitter, either TAPTR1 or TAPTR2 is used, depending on the portion of the serial interface TX RAM that is currently active. If the corresponding VTx or VRx bit is set, then:

- RAPTR1 contains the address of the currently active RX RAM entry. The serial interface RAM receive address block that is used is 0–63 and CRORA = 0 in the SISTR.
- RAPTR2 contains the address of the currently active RX RAM entry. The serial interface RAM receive address block that is used is 64–127 and CRORA = 1 in the SISTR.
- TAPTR1 contains the address of the currently active TX RAM entry. The serial interface RAM transmit address block that is used is 128–191 and CROTA = 0 in the SISTR.
- TAPTR2 contains the address of the currently active TX RAM entry. The serial interface RAM transmit address block that is used is 192–255 and CROTA = 1 in the SISTR.

16.7.6 IDL Interface Operation

The full-duplex ISDN interchip digital link (IDL) interface is used to connect a physical layer device to the MPC823. The basic and primary rate of the IDL bus is supported by the MPC823. In the basic rate of IDL, data on three channels (B1, B2, and D) is transferred in a 20-bit frame, providing 160kbps full-duplex bandwidth. The MPC823 is an IDL slave device that is clocked by the IDL bus master (physical layer device) and has separate receive and transmit sections. The MPC823 supports one IDL bus as illustrated in Figure 16-56.

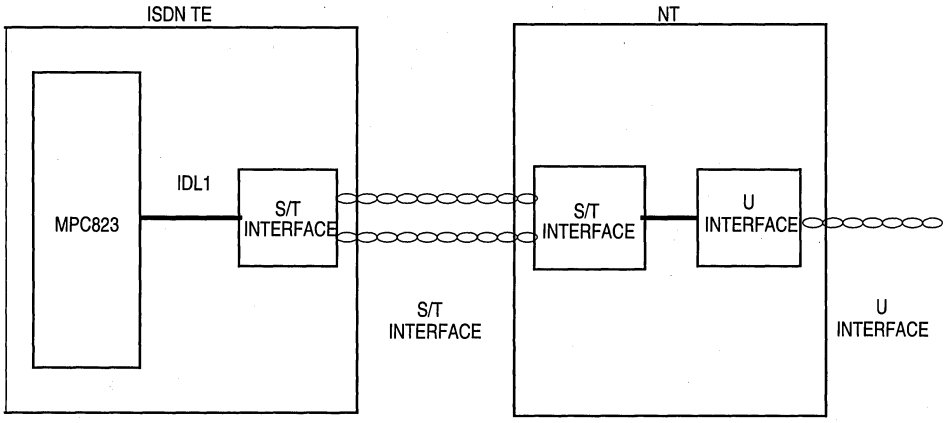


Figure 16-56. IDL Bus Application Example

16.7.6.1 IDL INTERFACE IMPLEMENTATION. The MPC823 can identify and support each IDL channel or it can output strobe lines for interfacing with devices that do not support the IDL bus. The IDL signals for each transmit and receive channel are as follows:

- L1RCLKA—IDL clock pin. Input to the MPC823.
- L1RSYNCA—IDL sync pin. Input to the MPC823. This signal indicates that the clock periods following the pulse designate the IDL frame.
- L1RXDA—IDL receive data pin. Input to the MPC823. Valid only for the bits that are supported by the IDL and ignored for other signals that may be present.
- L1TXDA—IDL transmit data pin. Output from the MPC823. Valid only for the bits that are supported by the IDL. Otherwise, it is three-stated.
- L1RQA—IDL request permission to transmit on the D channel. Output from the MPC823 on the L1RQA pin.
- L1GRA—IDL grant permission to transmit on the D channel. Input to the MPC823 on the L1TSYNCA pin.

The basic rate IDL bus has three channels:

- B1 is a 64kbps bearer channel
- B2 is a 64kbps bearer channel
- D is a 16kbps signaling channel

There are two definitions of the IDL bus frame structure—8 and 10 bits. The only difference between them is the channel order within the frame.

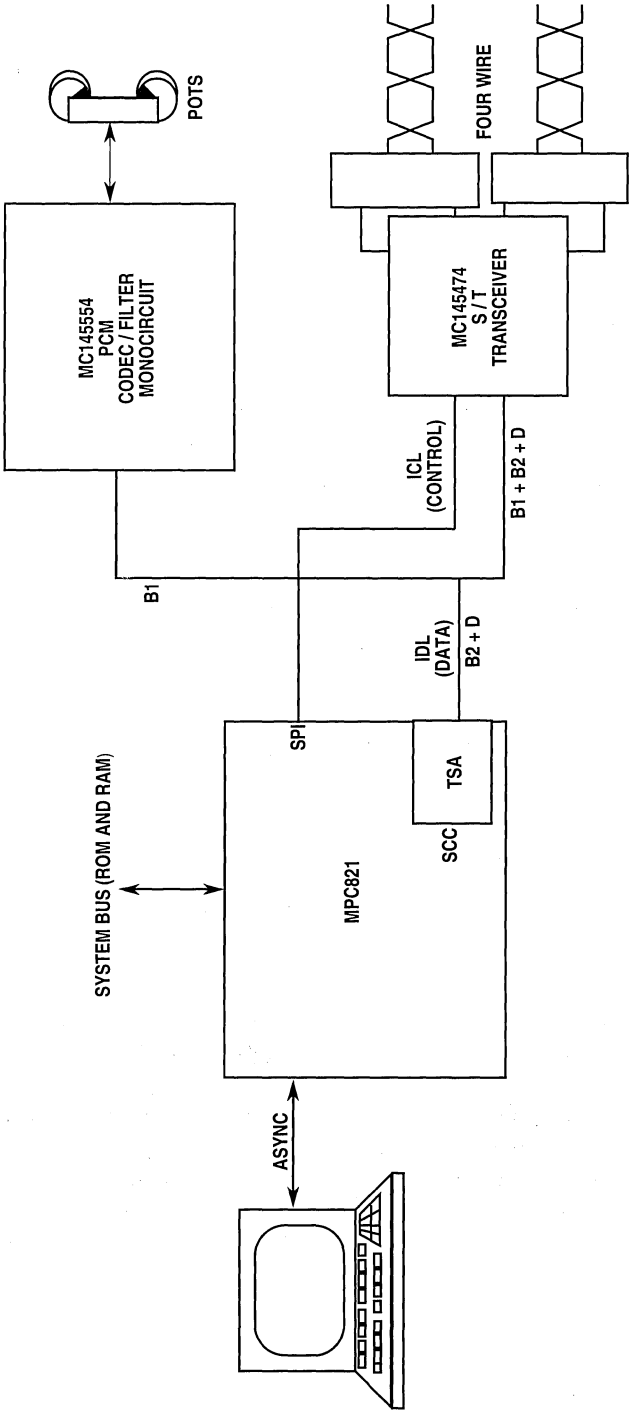
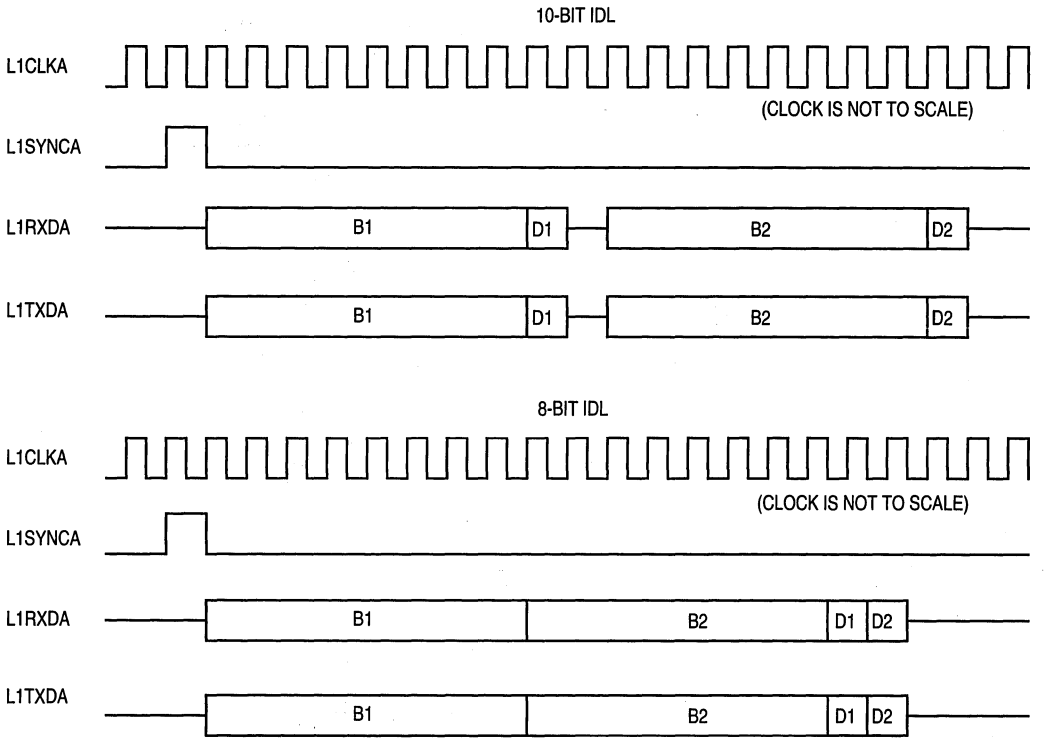


Figure 16-57. IDL Terminal Adaptor



NOTE: L1RQA AND L1GRA ARE NOT SHOWN.

Figure 16-58. IDL Bus Signals

Note: Previous versions of Motorola IDL-defined bit functions, called auxiliary (A) and maintenance (M), were eliminated from the IDL definition when it was concluded that the IDL control channel would be out-of-band. They were defined as a subset of the Motorola SPI format called serial control port (SCP). If you prefer to implement the A and M bit functions as originally defined, you can program the time-slot assigner to access these bits and route them transparently to a serial communication controller or serial management controller. To perform the out-of-band signaling required, use the MPC823 serial peripheral interface.

The MPC823 supports all channels of the IDL bus in the basic rate. Each bit in the IDL frame can be routed to every serial communication controller and serial management controller or they can assert a strobe output that supports an external device.

The MPC823 supports the request-grant method for contention detection on the D channel of the IDL basic rate and when the MPC823 has data to transmit on the D channel, it asserts the L1RQA pin. The physical layer device monitors the physical layer bus for activity on the D channel and indicates that the channel is free by asserting the L1GRA pin. The MPC823 samples the L1GRA pin when the IDL sync signal (L1RSYNCA) is asserted. If L1GRA is high (active), the MPC823 transmits the first zero of the opening flag in the first bit of the D channel. If a collision is detected on the D channel, the physical layer device negates L1GRA. The MPC823 then stops its transmission and retransmits the frame when L1GRA is reasserted. This procedure is handled automatically for the first two buffers of a frame.

For the primary rate IDL, the MPC823 supports up to four 8-bit channels in the frame, determined by the serial interface RAM programming. Additionally, the MPC823 can be used to assert strobes to support additional external IDL channels. The IDL interface supports the CCITT I.460 recommendation for data rate adaptation since it separately accesses each bit of the IDL bus. The current-route RAM specifies the bits that are supported by the IDL interface and the serial controller. The receiver only receives the bits that are enabled by the receiver route RAM. Otherwise, the transmitter only transmits the bits that are enabled by the transmitter route RAM and three-states the L1TXDA pin.

16.7.6.2 PROGRAMMING THE IDL INTERFACE. You can program the channels used for the IDL bus interface to the appropriate configuration. First, using the GMA bit, program the SIMODE register to the IDL grant mode for that channel. If the receive and transmit section are used for interfacing to the same IDL bus, using the CRTA bits you can internally connect the receive clock and sync signals to the serial interface RAM transmit section. However, the RAM section used for the IDL channels must be programmed to the preferred routing. For more information, see **Section 16.7.5 Serial Interface Programming Model**.

You should now define the IDL frame structure to be a 1-bit delay from frame sync to data, to falling edge sample sync, and the clock edge to transmit on the rising edge of the clock. Program the L1TXDA pin to be three-stated when inactive by using the parallel I/O open-drain register. To support the D channel, you must program the appropriate GMA bit in the SIMODE register and program the RAM entry to route data to that serial controller. The two definitions of IDL—8 and 10 bits—are only supported by modifying the serial interface RAM programming. In both cases, the L1GRA pin is sampled with the L1TSYNCA signal and transferred to the D channel serial communication controller as a grant indication. The same procedure is used to support an IDL bus in the second channel.

16.7.6.2.1 IDL Interface Programming Example. Using the example in **Section 16.7.6.1 IDL Interface Implementation** as a base model, the initialization sequence is as follows:

1. Assuming SMC1 is connected to the B1 channel, SMC2 is connected to the B2 channel, and SCC2 is connected to the D channel, program the serial interface RAM. Write all entries that are not used with 0x0001, set the LST bit, and disable the routing function.

ENTRY NUMBER	RAM WORD						
	SWTR	SSEL	CSEL	CNT	BYT	LST	DESCRIPTION
1	0	0000	101	0000	1	0	8 Bits SMC2 (B1)
2	0	0000	010	0000	0	0	1 Bit SCC2 (D)
3	0	0000	000	0000	0	0	1 Bit No Support
4	0	0000	110	0000	1	0	8 Bits SMC2 (B2)
5	0	0000	010	0000	0	1	1 Bit SCC2

Note: Since IDL requires the same routing for both receive and transmit, an exact duplicate of the above entries should be written to both the receive and transmit sections of the serial interface RAM beginning at serial interface RAM addresses 0 and 128, respectively.

2. SIMODE = 0x80008145. Only TDMa is used. The two SMCs are connected to the time-slot assigner.
3. SICR = 0x0000c000. SCC2 is connected to the time-slot assigner. SCC2 supports the grant mechanism since it is on the D channel.
4. PAODR bit 9 = 1. Configure L1TXDa to be an open-drain output.
5. PAPAR bits 9, 8, and 7 = 1. Configure L1TXDa, L1RXDa, and L1RCLKa.
6. PADIR bits 9 and 8 = 1. PADIR bit 7 = 0. Configure L1TXDa, L1RXDa, and L1RCLKa.
7. PCPAR bits 12, 5, and 11 = 1. Configure L1RQa, L1TSYNCa, and L1RSYNCa.
8. PCDIR bit 12 = 0. L1RQa is an input. L1TSYNCa performs the L1GRa function and is therefore an output, but it does not need to be configured with PCDIR bit 5 = 0. L1RSYNCa is an input, but it does not need to be configured with a PCDIR bit.
9. SIGMR = 0x04. Enable TDMa (one static TDM).
10. SICMR is not used.
11. SISTR and SIRP do not need to be read, but can be used for debugging information once the channels are enabled.
12. Enable SCC2 to HDLC operation (to handle the LAPD protocol of the D channel). Set SCC2 and SMC1 to transparent operation.

16.7.7 GCI Interface Operation

The normal mode of the general circuit interface (GCI) and SCIT are fully supported by the MPC823. It also supports the D channel access control in S/T interface terminals by using the command/indication (C/I) channel for that function.

The GCI bus consists of four lines—two data lines, a clock, and a frame synchronization line. Usually, an 8kHz frame structure defines the various channels within the 256kbps data rate. The MPC823 supports an independent GCI bus and has independent receive and transmit sections. The interface can also be used in a multiplexed frame structure on which up to eight physical layer devices multiplex their GCI channels. In this mode, the data rate would be 2,048kbps. In the GCI bus, the clock rate is twice the data rate. The serial interface divides the input clock by two to produce the data clock. The MPC823 also has data strobe lines that are used as an interface for devices that do not support the GCI bus. The GCI signals for each transmit and receive channel are as follows:

- L1RSYNCA—Used as a GCI sync signal. Input to the MPC823. This signal indicates that the clock periods following the pulse designate the GCI frame.
- L1RCLKA—Used as a GCI clock. Input to the MPC823. The L1RCLKA signal is twice the data clock.
- L1RXDA—Used as a GCI receive data. Input to the MPC823.
- L1TXDA—Used as a GCI transmit data. Open-drain output. Valid only for the bits that are supported by the IDL. Otherwise, it is three-stated.

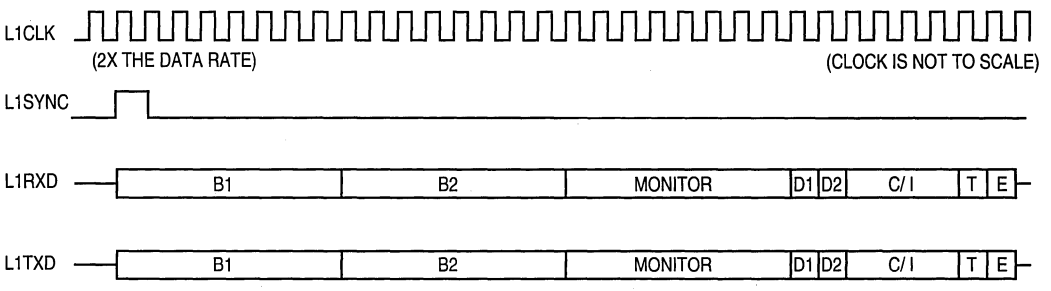


Figure 16-59. GCI Bus Signals

In addition to the 144kbps ISDN 2B+D channels, the GCI provides five channels for maintenance and control functions:

- B1 is a 64kbps bearer channel
- B2 is a 64kbps bearer channel
- M is a 64kbps monitor (M) channel
- D is a 16kbps signaling channel
- C/I is a 48kbps C/I channel (includes Tand E bits)

The M channel is used to transfer data between layer 1 devices and the control unit (the core) and the C/I channel is used to control activation/deactivation procedures or to switch test loops by the control unit. The M and C/I channels of the GCI bus should be routed to SMC1 or SMC2, which have modes to support the channel protocols. The MPC823 can support any channel of the GCI bus in the primary rate by modifying the serial interface RAM programming.

The GCI supports the CCITT I.460 recommendation as a method for data rate adaptation since it can access each bit of the GCI separately. The current-route RAM specifies which the bits that are supported by the interface and serial controller. The receiver only receives the bits that are enabled by the serial interface RAM and the transmitter only transmits the bits that are enabled by the serial interface RAM and does not drive L1TXDA. Otherwise, L1TXDA is an open-drain output and should be externally pulled high.

The MPC823 supports contention detection on the D channel of the SCIT bus. When the MPC823 has data to transmit on the D channel, it checks a SCIT bus bit that is marked with a special route code (usually, bit 4 of C/I channel 2). The physical layer device monitors the physical layer bus for activity on the D channel and indicates on this bit that the channel is free. If a collision is detected on the D channel, the physical layer device sets bit 4 of C/I channel 2 to logic high. The MPC823 then aborts its transmission and retransmits the frame when this bit is set again. This procedure is automatically handled for the first two buffers of a frame.

16.7.7.1 GCI ACTIVATION/DEACTIVATION PROCEDURE. In the deactivated state, the clock pulse is disabled and the data line is at a logic one. The layer 1 device activates the MPC823 by enabling the clock pulses and sending an indication to C/I channel 0. Using a maskable interrupt, the MPC823 lets the core know that a valid indication has been received in the SMC receive buffer descriptor.

When the core activates the line, the data output of L1TXDA is programmed to zero by setting the STZA bit in the SIMODE register. Code 0 (command timing TIM) is transmitted on C/I channel 0 to the layer 1 device until the STZA bit is reset. The physical layer device resumes the clock pulses and gives an indication in C/I channel 0. The core should reset the STZA bit to enable data output.

16.7.7.2 PROGRAMMING THE GCI INTERFACE. There are two modes of the GCI interface—normal and SCIT.

16.7.7.2.1 Normal Mode. You can program and configure the channels used for the GCI bus interface. First, program the SIMODE register to the GCI/SCIT mode for that channel. This mode defines the sync pulse to GCI sync for framing and data clock as one-half the input clock rate. You can program more than one channel to interface to the GCI bus. Also, if the receive and transmit section are used for interfacing the same GCI bus, using the CRTA bit you should internally connect the receive clock and sync signals to the serial interface RAM transmit section. Then you should define the GCI frame routing and strobe select using the serial interface RAM.

When the receive and transmit sections use the same clock and sync signals, these sections should be programmed to the same configuration. Also, the L1TXDA pin in the I/O register should be programmed to be an open-drain output. To support the monitor and C/I channels in GCI, those channels should be routed to one of the serial management controllers. To support the D channel when there is no possibility of collision, you should clear the GRA bit corresponding to the serial communication controller that supports the D channel in the SIMODE register.

16.7.7.2.2 SCIT Mode. To interface with the GCI/SCIT bus, the SIMODE register must be programmed to GCI/SCIT mode. The serial interface RAM is programmed to support a 96-bit frame length and the frame sync is programmed to the GCI sync pulse. Generally, the SCIT bus supports the D channel access collision mechanism. For this purpose, you should program the receive and transmit sections to use the same clock and sync signals with the CRTA bit and program the GRA bit to transfer the D channel grant to the serial communication controller that supports this channel. The received bit should be marked by programming the channel select bits of the serial interface RAM to 111 for an internal assertion of a strobe on this bit. This bit is sampled by the serial interface and transferred to the D channel serial communication controller as the grant. The bit is generally bit 4 of the C/I in channel 2 of GCI, but any other bit can be selected using the serial interface RAM.

16.7.7.3 GCI INTERFACE PROGRAMMING EXAMPLE. Assuming SCC2 is connected to the D channel, SMC2 to the B1 channel, and SMC1 to the C/I channels, the initialization sequence is as follows:

1. Program the serial interface RAM. Write all entries that are not used with 0x0001, set the LST bit, and disable the routing function.

ENTRY NUMBER	RAM WORD						
	SWTR	SSEL	CSEL	CNT	BYT	LST	DESCRIPTION
1	0	0000	110	0000	1	0	8 Bits SMC2 (B1)
2	0	0001	000	0000	1	0	Ext Device (B2)
3	0	0000	101	0000	1	0	8 Bits SMC1 (M)
4	0	0000	010	0001	0	0	2 Bits SCC2 (D)
5	0	0000	101	0101	0	0	6 Bits SMC1 (I+A+E)
6	0	0000	000	0110	1	0	Skip 7 Bytes
7	0	0000	000	0001	0	0	Skip 2 Bits
8	0	0000	111	0000	0	1	D Grant Bit

Note: Since GCI requires the same routing for both receive and transmit, an exact duplicate of the above entries should be written to both the receive and transmit sections of the serial interface RAM beginning at addresses 0 and 128, respectively.

2. SIMODE = 0x0000c000. SCC2 is connected to the time-slot assigner. SCC2 supports the grant mechanism since it is on the D channel.

Note: If SCIT mode is not used, delete the last three entries of the serial interface RAM and set the LST bit in the new last entry.

3. SICR = 0x000040C0. SCC2 is connected to the time-slot assigner.
4. PAODR bit 9 = 1. Configure L1TXDa to be an open-drain output.
5. PAPAR bits 9, 8, and 7= 1. Configure L1TXDa, L1RXDa, and L1RCLKa.
6. PADIR bits 9 and 8 = 1.PADIR bit 7 = 0. Configure L1TXDa, L1RXDa, and L1RCLKa.
7. PCPAR bit 4 = 1. Configure L1RSYNCa.
8. SIGMR = 0x04.Enable TDMa (one static TDM).
9. SICMR is not used.
10. SISTR and SIRP do not need to be read, but can be used for debugging information once the channels are enabled.
11. Enable SCC2 for HDLC operation (to handle the LAPD protocol of the D channel). Set SMC1 for SCIT operation. Set SMC2 for transparent operation.

16.7.8 Nonmultiplexed Serial Interface Configuration

The serial interface supports a nonmultiplexed serial interface (NMSI) mode for the serial communication controller and serial management controllers. The decision of whether to connect SCC2 to the NMSI is made in the SICR and the decision of whether to connect a serial management controller to the NMSI is made in the SIMODE register. The serial communication controller or serial management controllers can be connected to the NMSI, regardless of the other channels connected to a TDM channel. You should keep in mind, however, that NMSI pins can be multiplexed with other functions at the parallel I/O lines. Therefore, if a combination of the TDMA and NMSI channels is used, you should consult the MPC823 pinout in **Section 2 External Signals** to decide which serial communication controller or serial management controller to connect and where to connect them.

The clocks that are provided to the universal serial bus, serial communication controller, and serial management controllers are derived from six sources—four internal baud rate generators and four external CLK pins. There are two main advantages to this bank-of-clocks approach. First, a universal serial bus, serial communication controller, or serial management controller is not forced to choose its clock from a predefined pin or baud rate generator, which adds flexibility to the pinout mapping strategy. Second, if a group of SMC receivers and transmitters need the same clock rate they can share the same pin, which leaves other pins available for other functions and minimizes the potential skew between multiple clock sources.

The four baud rate generators also make their clocks available to external logic, regardless of whether they are being used by a serial communication controller or serial management controller. The BRGOx pins are multiplexed with other functions, so all BRGO*x* pins may not always be available. For more information, see the pinout in **Section 2 External Signals**. The bank-of-clocks mapping has one restriction—the SMC transmitter must have the same clock source as the receiver when connected to the NMSI pins.

Once the clock source is selected, the clock is given an internal name. For the serial communication controller, the name is RCLK2 and TCLK2 and for the serial management controllers, the name is SMCLKx. These internal names are used only in NMSI mode to specify the clock that is sent to the serial communication controller or serial management controller. These names do not correspond to any pins on the MPC823.

Note: The internal RCLK2 and TCLK2 can be used as inputs to the DPLL unit, which is inside the serial communication controller. Thus, the RCLK2 and TCLK2 signals are not always required to reflect the actual bit rate on the line.

The pins available to the universal serial bus, serial communication controller, and serial management controllers in NMSI mode are illustrated in Figure 16-60.

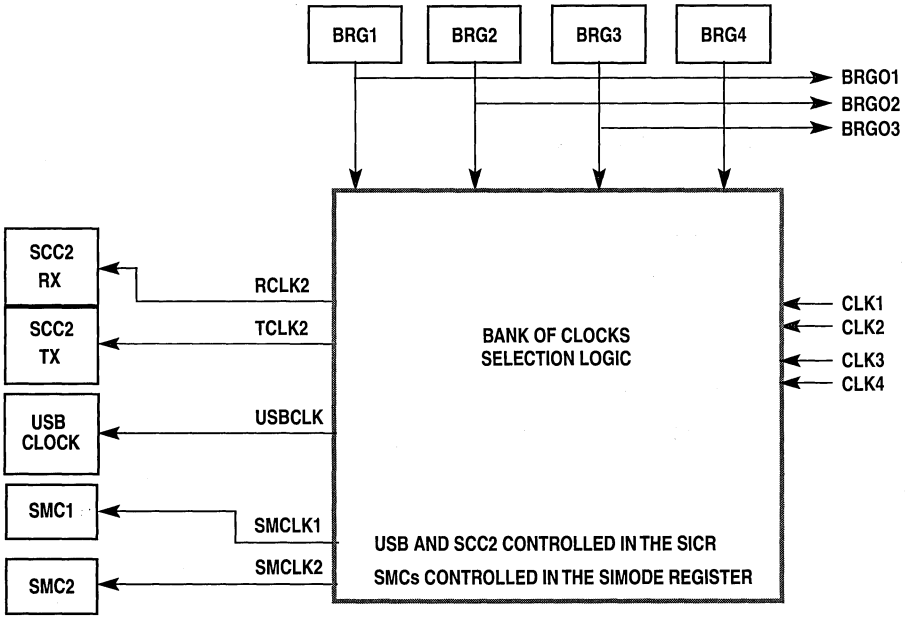


Figure 16-60. Bank of Clocks

USB in NMSI mode has its own set of modem control pins:

- USBRXD
- USBRXP
- USBTXP
- USBCLK ← BRG1–BRG4, CLK1–CLK4
- USBRXN
- USBOE

SCC2 in NMSI mode has its own set of modem control pins:

- TXD2
- RXD2
- TCLK2 ← BRG1–BRG4, CLK1–CLK4
- RCLK2 ← BRG1–BRG4, CLK1–CLK4
- RTS2
- CTS2
- CD2

SMC1 in NMSI mode has its own set of modem control pins:

- SMTXD1
- SMRXD1
- SMCLK1 ← BRG1–BRG4, CLK1–CLK4
- SMSYN1 (used only in the totally transparent protocol)

SMC2 in NMSI mode has its own set of modem control pins:

- SMTXD2
- SMRXD2
- SMCLK2 ← BRG1–BRG4, CLK1–CLK4
- SMSYN2 (used only in the totally transparent protocol)

A universal serial bus, serial communication controller, or serial management controller that requires fewer pins than those listed above can use that pin for another function or configure that pin as a parallel I/O pin.

16.8 THE BAUD RATE GENERATORS

The communication processor module contains four, independent, identical baud rate generators (BRGs) that can be used with the universal serial bus, serial communication controller, and serial management controllers. The clocks produced in the baud rate generator are sent to the bank-of-clocks selection logic, where they can be routed to the universal serial bus, serial communication controller, or serial management controllers. In addition, the output of the baud rate generator can be routed to a pin to be used externally. The following is a list of baud rate generators' main features:

- Four independent and identical baud rate generators. If you are using Mask Revision Base #F98S, there are only two baud rate generators.
- On-the-fly changes are allowed.
- Each baud rate generator can be routed to the universal serial bus, serial communication controller, or serial management controllers.
- A 16× divider option allows low baud rates at high system frequencies
- Each baud rate generator contains an autobaud support option.
- Three of the baud rate generator outputs can be routed to the BRGOx pin.

The baud rate generator block diagram is illustrated in Figure 16-61.

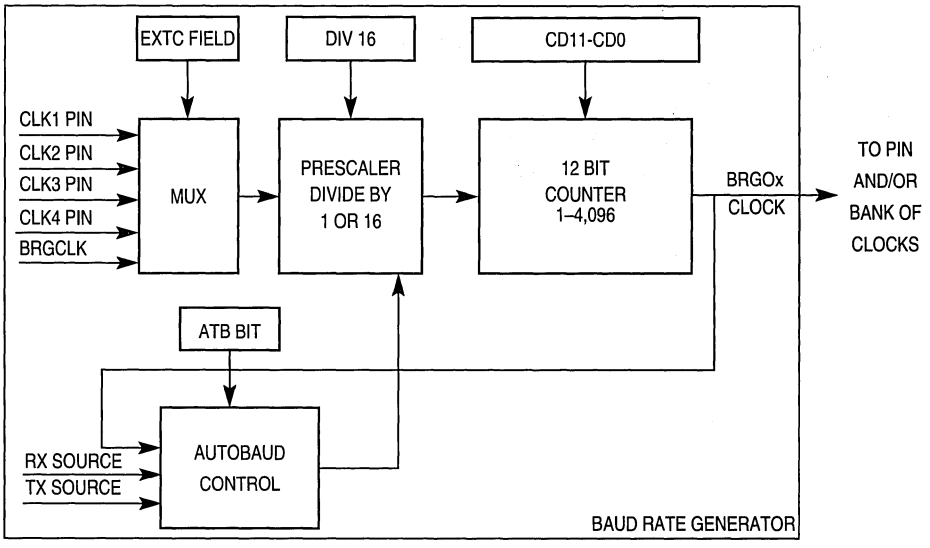


Figure 16-61. Baud Rate Generator Block Diagram

The clock input to the prescaler can be selected by the EXTC field in the BRGCx register to originate in one of five sources—BRGCLK, CLK1, CLK2, CLK3, or CLK4.

The BRGCLK is generated in the MPC823 clock synthesizer specifically for the four baud rate generators, serial peripheral interface, and I^2C internal baud rate generator. You can also configure the CLK1, CLK2, CLK3, or CLK4 pins to be the clock source. An external pin allows flexible baud rate frequency generation, regardless of the system frequency. Additionally, the CLK1, CLK2, CLK3, or CLK4 pins allow a single external frequency to become the input clock for multiple baud rate generators. The clock signals on the CLK1, CLK2, CLK3, or CLK4 pins are not synchronized internally prior to being used by the baud rate generator.

Next, the baud rate generator provides a divide-by-16 option before the clock reaches the prescaler. This option is chosen by the DIV16 bit. The clock is then divided into the prescaler by a maximum of 4,096. This input clock divide ratio can be programmed on-the-fly. However, on-the-fly baud rate generator changes should not occur within a shorter time than the period of two baud rate generator input clocks.

Note: Four baud rate generators are available for the SCC and SMCs. However, BRGO4 cannot be output externally.

The output of the prescaler is sent internally to the bank of clocks and can also be output externally on the BRGOx pins of either the port A or port B parallel I/O. One BRGOx pin is an output from the corresponding baud rate generator. If the baud rate generator divides the clock by an even value, the transitions of the BRGOx pin always occur on the falling edge of the input clock to the baud rate generator. If the baud rate generator is programmed to an odd value, the transitions alternate between the falling and rising edges of the input clock. Additionally, the output of the baud rate generator can be sent to the autobaud control block.

16.8.1 Autobaud Operation

Using the autobaud process, a UART determines the baud rate of its received character stream by examining the pattern received and the timing information of that pattern. The MPC823 baud rate generators have a built-in autobaud control function that automatically measures the length of a start bit and modifies the baud rate accordingly. If the ATB bit in the BRGCx register is set, the autobaud control block starts searching for a low level on the RXD2 signal, which it assumes is the beginning of a start bit and begins counting the start bit length. During this time, the BRG output clock toggles for 16 BRG clock cycles at the BRG input clock rate and then stops with the BRGOx output clock in the low state.

After the RXD2 signal changes back to the high level, the autobaud control block rewrites the CD and DIV16 bits in the BRGCx register to the divide ratio it found. Due to a measurement error that occurs at high baud rates, this divide ratio written by the autobaud controller may not be the precise, baud rate you prefer (56,600 could be the resulting baud rate, rather than 57,600). Thus, an interrupt is provided for you in the SCCE–UART register to signify that the BRGCx register was rewritten by the autobaud controller. When this interrupt is recognized, you should rewrite the BRGCx with the value you prefer. It is recommended that this be done as quickly as possible (even prior to the first character being fully received) to ensure that all characters are recognized correctly by the UART.

Once a full character is received, you can check the software to verify that the received character matches a predefined value (such as “a” or “A”). The software should then check for other characters (such as “t” or “T”) and program the serial communication controller to the preferred parity mode. You can change the parity mode in the SCC2 UART mode register (PSMR–SCC2 UART), which is in **Section 16.9.3 Protocol-Specific Mode Register**.

Note: The serial communication controller associated with this baud rate generator must be programmed to UART mode and have the TDCR and RDCR fields in the GSMR_L set to the 16× option for the autobaud function to operate correctly. Input frequencies such as 1.32MHz, 3.68MHz, 7.36 MHz, and 14.72MHz should be used. For autobaud to operate successfully, the serial communication controller must be connected to the baud rate generator. For instance, SCC2 must be clocked by BRG2 to successfully perform the autobaud function. Also, for the serial communication controller to correctly detect an autobaud lock and generate an interrupt, it must receive three full RX clocks from the baud rate generator before the autobaud process begins. To do this, set the ATB bit to 0 in the BRGCx register and enable the BRGx receive clock to the highest frequency. Immediately prior to the start of the autobaud process (after device initialization), set the ATB bit to 1.

16.8.2 Baud Rate Generator Configuration Registers

The 32-bit, memory-mapped baud rate generator configuration registers (BRGC1–4) are are used to configure the baud rate generators. At reset, the baud rate generators are disabled and the BRGOx output clock is high. These registers can be written at any time and you do not need to disable any internal or external devices connected to the BRGOx output clock. Changes to the baud rate generators occur at the end of the next BRG clock cycle (no spikes occur on the BRGOx output clock).

BRGC1–BRGC4

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED														RST	EN
RESET	0														0	0
R/W	R/W														R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x9F0 (BRGC1), 0x9F4 (BRGC2), 0x9F8 (BRGC3), 0x9FC (BRGC4)															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	EXTC		ATB	CD												DIV16
RESET	0		0	0												0
R/W	R/W		R/W	R/W												R/W
ADDR	(IMMR & 0xFFFF0000) + 0x9F2 (BRGC1), 0x9F6 (BRGC2), 0x9FA (BRGC3), 0x9FE (BRGC4)															

Bits 0–13—Reserved

These bits are reserved and should be set to 0.

RST—Reset Baud Rate Generator

This bit performs a software reset of the baud rate generator identical to that of an external reset. A reset disables the baud rate generator and sets the BRGOx output clock. This can only be seen externally if the BRGOx function is enabled to reach the corresponding port A or B parallel I/O pin.

0 = Enable the baud rate generator.
1 = Reset the baud rate generator (software reset).

EN—Enable Baud Rate Generator Count

This bit is used to dynamically stop the baud rate generator from counting, which may be useful for low-power modes.

0 = Stop all clocks to the baud rate generator.
1 = Enable clocks to the baud rate generator.

EXTC—External Clock Source

This field selects the baud rate generator input clock from the internal BRGCLK or one of three external pins.

00 = The baud rate generator input clock comes from the BRGCLK (internal clock generated by the clock synthesizer in the system interface unit).
01 = The baud rate generator input clock comes from the CLK2 pin.
10 = The baud rate generator input clock comes from the CLK4 pin.
11 = Reserved.

ATB—Autobaud

When set, this bit selects autobaud operation of the baud rate generator on the corresponding RXD2 pin.

0 = Normal operation of the baud rate generator.
1 = When RXD2 goes low, the baud rate generator determines the length of the start bit and synchronizes the baud rate generator to the actual baud rate.

Note: The ATB bit must remain clear (0) until the serial communication controller receives the three RX clocks. Then you must set this bit to one to obtain the correct baud rate. Once the baud rate is obtained and locked, it is indicated by setting the AB bit in the UART SCCE register in **Section 16.9.8.1 SCC2 Event Register**. This bit may only be set for BRG2.

CD—Clock Divider

This field and the prescaler determines the baud rate generator output clock rate. They are used to preset a 12-bit counter that is decremented at the prescaler output rate, but the counter is inaccessible to you. When the counter reaches zero, it is reloaded from the clock divider bits. Thus, a value of 0xFFF in CD0–CD11 produces the minimum clock rate (divide by 4,096), and a value of 0x0000 produces the maximum clock rate (divide by 1). Even when dividing by an odd number, the counter ensures a 50% duty-cycle by asserting the terminal count once on clock low and next on clock high. The terminal count signals counter expiration and toggles the clock.

DIV16—BRG Clock Prescaler Divide by 16

0 = Divide by 1 for the clock divider input.
1 = Divide-by-16 prescaler enabled for the clock divider output.

16.8.3 UART Baud Rate Examples

For synchronous communication using the internal baud rate generator, the BRGOx output clock must never be faster than the system frequency divided by 2. So, with a 25MHz system frequency, the maximum BRGOx output clock rate is 12.5MHz. You should program the UART to 16× oversampling when using the serial communication controller as a UART. On the MPC823, 8× and 32× options are also available. Assuming 16× oversampling is chosen in the UART, a data rate of 25MHz ÷ 16 = 1.5625Mb/sec is the maximum possible UART speed. Putting this together, the following formula should be used to calculate the bit rate based on a particular baud rate generator configuration for a UART:

async baud rate = (BRGCLK, CLK1, CLK2, CLK3 or CLK4) ÷ (clock divider + 1) ÷ (1 or 16 depending on the DIV16 bit) ÷ (8, 16, or 32 according to the RDCR and TDCR fields in the GSMR_L register).

The following table lists typical bit rates of asynchronous communication. The internal clock rate is assumed to be 16× the baud rate.

Table 16-23. Typical Baud Rates of Asynchronous Communication

BAUD RATES	MPC823 SYSTEM FREQUENCY (MHz)								
	20			25			24.5760		
	DIV16	DIV	ACTUAL FREQUENCY	DIV16	DIV	ACTUAL FREQUENCY	DIV16	DIV	ACTUAL FREQUENCY
50	1	1561	50.02	1	1952	50	1	1919	50
75	1	1040	75.05	1	1301	75	1	1279	75
150	1	520	149.954	1	650	150	1	639	150
300	1	259	300.48	1	324	300.5	1	319	300
600	0	2082	600.09	0	2603	600	0	2559	600
1200	0	1040	1200.7	0	1301	1200	0	1279	1200
2400	0	520	2399.2	0	650	2400.1	0	639	2400
4800	0	259	4807.7	0	324	4807.69	0	319	4800
9600	0	129	9615.4	0	162	9585.9	0	159	9600
19200	0	64	19231	0	80	19290	0	79	19200
38400	0	32	37879	0	40	38109	0	39	38400
57600	0	21	56818	0	26	57870	0	26	56889
115200	0	10	113636	0	13	111607	0	12	118154

NOTE: All values are decimal.

For synchronous communication, the internal clock is identical to the baud rate output. To get the preferred rate, you can select the appropriate system clock according to the following equation:

sync baud rate = (BRGCLK, CLK1, CLK2, CLK3 or CLK4) ÷ (clock divider + 1) ÷ (1 or 16 according to the DIV16 bit)

To get a rate of 64kbps, the system clock can be 24.96MHz, DIV16 = 0, and the clock divider = 389.

16.9 THE SERIAL COMMUNICATION CONTROLLER

The MPC823 has a serial communication controller (SCC2) that can be configured independently to implement different protocols. It can be used to implement bridging functions, routers, gateways, and interface with a wide variety of standard WANs, LANs, and proprietary networks. The serial communication controller has many physical interface options such as interfacing to TDM buses, ISDN buses, and standard modem interfaces.

The serial communication controller does not include the physical interface, but it is the logic that formats and manipulates the data obtained from the physical interface. That is why the serial interface is described in a different section. The protocol you choose is independent of your physical interface preference. The serial communication controller is described in terms of the protocol that you choose to run. When a serial communication controller is programmed to implement a certain protocol or mode, a certain level of functionality is associated with that protocol. For most protocols, this corresponds to the different parts of the link layer. Many functions of the serial communication controller are common to all of the following protocols:

- UART controller
- HDLC controller
- HDLC bus controller
- AppleTalk/LocalTalk controller
- Asynchronous HDLC controller
- IrDA controller
- Transparent controller
- Ethernet controller

Although the selected protocol usually applies to both the SCC2 transmitter and receiver, the serial communication controller can run half of the serial communication controller with transparent operation while the other half runs the standard protocol.

Each of the internal clocks for the serial communication controller can be programmed with either an external or internal source. The internal clocks originate from one of four baud rate generators or one of four external clock pins. These clocks can be as fast as a 1:2 ratio of a 12.5MHz system clock. However, the serial communication controller's ability to support a sustained bitstream depends on the protocol as well as other factors. Associated with the serial communication controller is a digital phase-locked loop (DPLL) for external clock recovery, which includes NRZ, NRZI, FM0, FM1, Manchester, and Differential Manchester. The DPLL can be configured to NRZ operation to pass the clocks and data to or from the serial communication controller without modifying it.

The RISC microcontroller is responsible for selecting and controlling the various ports and controllers of the communication processor module. For the serial communication controller command set, refer to Table 16-2.

The serial communication controller can be connected to its own set of pins on the MPC823. This configuration is called the nonmultiplexed serial interface (NMSI) and is described in **Section 16.7 The Serial Interface with Time-Slot Assigner**. In this configuration, the serial communication controller can support the standard modem interface signals—$\overline{\text{RTS}}$, $\overline{\text{CTS}}$, and $\overline{\text{CD}}$—through the port B or C pins and the CPM interrupt controller. Additional handshake signals can be supported with additional parallel I/O lines. The serial communication controller block diagram is illustrated in Figure 16-62.

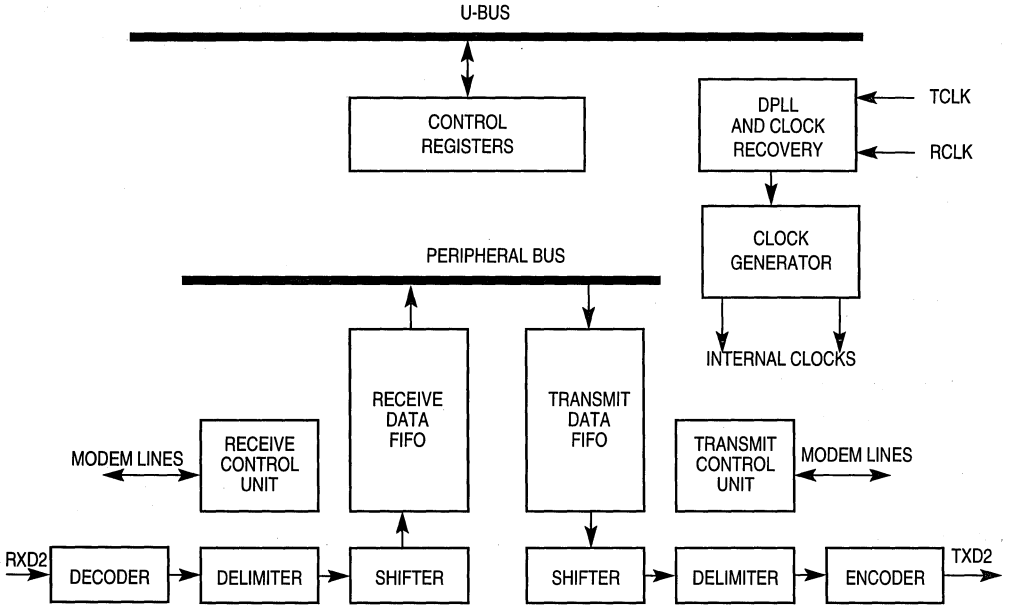


Figure 16-62. Serial Communication Controller Block Diagram

16.9.1 Features

The following is a list of the serial communication controller's main features:

- Implements the HDLC, HDLC bus, asynchronous HDLC, synchronous start/stop, asynchronous start/stop, AppleTalk, Transparent, and Ethernet protocols
- Supports full 10Mbps Ethernet/IEEE 802.3
- A single HDLC or transparent channel can be supported at 8Mbps (full duplex)
- Maximum clocking rates of 12.5MHz at 25MHz
- DPLL circuitry for clock recovery with NRZ, NRZI, FM0, FM1, Manchester, and Differential Manchester (also known as Differential Bi-phase-L)
- Clocks can be derived from a baud rate generator, an external pin, or DPLL; data clock can be as high as 3.125MHz with a 25MHz clock
- Supports automatic control of the $\overline{\text{RTS}}$, $\overline{\text{CTS}}$, and $\overline{\text{CD}}$ modem signals
- Multibuffer data structure for receive and transmit (up to 512 buffer descriptors can be partitioned in any way)
- Deep FIFOs
- Transmit-on-demand feature decreases time-to-frame transmission
- Low FIFO latency option for transmit and receive in character-oriented and totally transparent protocols
- Frame preamble options
- Full-duplex operation
- Fully transparent option for receiver/transmitter while another protocol executes on the transmitter/receiver
- Echo and local loopback modes for testing

Note: The serial communication controller is the same on both the MPC821 and MPC823, except that the MPC823's SCC2 supports a wider range of IrDA signaling rates. The register control bits have been removed so the SIR and IRP bits of the GSMR_H and GSMR_L are reserved on the MPC823. However, the MPC823 does not support the BISYNC protocol or the external content-addressable memory (CAM) for the Ethernet/IEEE 802.3 protocol.

16.9.2 The General SCC2 Mode Registers

The serial communication controller contains two high and low read/write general SCC2 mode registers (GSMR_H and GSMR_L) that define all the options common to the serial communication controller, regardless of the protocol. These registers are cleared at reset and since they are 64 bits in length, they are accessed as GSMR_L and GSMR_H. GSMR_L contains the first (low-order) 32 bits of the GSMR and GSMR_H contains the last 32 bits.

GSMR_H

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															GDE
RESET	0															0
R/W	R/W															R/W
ADDR	(IMMR & 0xFFF0000) + 0xA24															

BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	TCRC		REVD	TRX	TTX	CDP	CTSP	CDS	CTSS	TFL	RFW	TXSY	SYNL		RTSM	RSYN
RESET	0		0	0	0	0	0	0	0	0	0	0	0		0	0
R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W
ADDR	(IMMR & 0xFFF0000) + 0xA26															

Bits 0–14—Reserved

These bits are reserved and should be set to 0.

GDE—Glitch Detect Enable

This bit determines whether the serial communication controller will search for glitches on the external receive and transmit serial clock lines provided. If this feature is enabled, the presence of a glitch is reported in the SCC2 event register. Whether or not the GDE bit is set, the serial communication controller always attempts to clean up the clocks that it uses internally, via a Schmitt trigger on the input lines.

0 = No glitch detection is performed. This option should be chosen if the external serial clock exceeds the limits of the glitch detection logic (6.25MHz assuming a 25MHz system clock). This option should also be chosen if the SCC2 clock is provided from one of the internal baud rate generators. Lastly, this option should be chosen if external clocks are used and it is more important to minimize power consumption than to watch for glitches.

1 = Glitch detection is performed with a maskable interrupt generated in the SCC2 event register.

TCRC—Transparent CRC (Totally Transparent Mode Only)

This field selects the type of frame checking that is provided on the transparent channels of the serial communication controller (either the receiver, transmitter, or both, as defined by the TTX and TRX bits). Although this configuration selects a frame check type, the actual decision to send the frame check is made in the TX buffer descriptor. Thus, it is not required to send a frame check in transparent mode. If a frame check is not used, you can simply ignore the frame check errors that are generated on the receiver.

00 = 16-bit CCITT CRC (HDLC). (X16 + X12 + X5 + 1).
01 = CRC16. (X16 + X15 + X2 + 1).
10 = 32-bit CCITT CRC (Ethernet, HDLC, and High-Speed IrDA). (X32 + X26 + X23 + X22 + X16 + X12 + X11 + X10 + X8 + X7 + X5 + X4 + X2 + X1 +1). Also called CRC32.
11 = Reserved.

REVD—Reverse Data (Transparent Mode Only)

0 = Normal operation.
1 = When set, this bit causes the totally transparent channels on this serial communication controller (either the receiver, transmitter, or both) to reverse the bit order, transmitting the MSB of each octet first.

TRX—Transparent Receiver

The serial communication controller offers totally transparent operation. However, to increase flexibility, totally transparent operation is not configured with the MODE field of the GSMR_L, but with the TTX and TRX bits. This gives you the opportunity to implement unique applications, such as configuring an SCC2 transmitter to a UART and configuring the receiver to a totally transparent operation. To do this, set MODE = UART, TTX = 0, and TRX = 1.

0 = Normal operation.
1 = The receiver operates in totally transparent mode, regardless of the protocol selected for the transmitter in the MODE field of the GSMR_L.

Note: Full-duplex transparent operation for a serial communication controller is obtained by setting both TTX and TRX. A serial communication controller cannot operate with Ethernet on its transmitter or while transparent operation is on its receiver. In other words, if the MODE field is set for Ethernet, then TTX must equal TRX or erratic operation will occur.

TTX—Transparent Transmitter

The serial communication controller offers totally transparent operation. However, to increase flexibility, totally transparent operation is not configured with the MODE field, but with the TTX and TRX bits. This gives you the opportunity to implement unique applications, such as configuring an SCC2 receiver to an HDLC and configuring a transmitter to a totally transparent operation. To do this, set MODE = HDLC, TTX = 1, and TRX = 0.

0 = Normal operation.
1 = The transmitter operates in totally transparent mode, regardless of the protocol selected for the receiver in the MODE field of the GSMR_L.

Note: Full-duplex transparent operation for a serial communication controller is obtained by setting both TTX and TRX. The serial communication controller cannot operate with Ethernet on its receiver or while transparent operation is on its transmitter. In other words, if the MODE field is set to Ethernet, then TTX must equal TRX or erratic operation will occur.

CDP—Carrier-Detect Pulse

This bit must be set if the serial communication controller is used in the time-slot assigner.

0 = Normal operation (envelope mode). The $\overline{CD}$ signal should envelope the frame. Negating $\overline{CD}$ while receiving data causes a $\overline{CD}$ lost error.
1 = Pulse mode. Once the $\overline{CD}$ signal is asserted, synchronization has been achieved and further transitions of $\overline{CD}$ have no effect on reception.

CTSP—$\overline{CTS}$ Pulse

0 = Normal operation (envelope mode). The $\overline{CTS}$ signal should envelope the frame. Negating $\overline{CTS}$ while transmitting data causes a $\overline{CTS}$ lost error.
1 = Pulse mode. Once the $\overline{CTS}$ signal is asserted, synchronization has been achieved and further transitions of $\overline{CTS}$ have no effect on transmission.

CDS—Carrier-Detect Sampling

0 = The $\overline{CD}$ signal is assumed to be asynchronous with the data. It is internally synchronized by the serial communication controller and then data is received.
1 = The $\overline{CD}$ signal is assumed to be synchronous with the data, which speeds up operation. In this mode, $\overline{CD}$ must transition while the receive clock is in the low state. As soon as $\overline{CD}$ is low, data is received. This mode is especially useful when connecting an MPC823 in transparent mode because it allows the $\overline{RTS}$ signal of one MPC823 to be directly connected to the $\overline{CD}$ signal of another MPC823.

CTSS—$\overline{\text{CTS}}$ Sampling

0 = The $\overline{\text{CTS}}$ signal is assumed to be asynchronous with the data. It is internally synchronized by the serial communication controller and data is then transmitted after several serial clock delays.

1 = The $\overline{\text{CTS}}$ signal is assumed to be synchronous with the data, which speeds up operation. In this mode, $\overline{\text{CTS}}$ must transition while the transmit clock is in the low state. As soon as $\overline{\text{CTS}}$ is low, data immediately begins transmission. This mode is especially useful when connecting an MPC823 in transparent mode since it allows the $\overline{\text{RTS}}$ signal of one MPC823 to be directly connected to the $\overline{\text{CTS}}$ signal of another MPC823.

TFL—Transmit FIFO Length

0 = Normal operation. The transmit FIFO is 32 bytes for the serial communication controller.

1 = The transmit FIFO is 1 byte and can be used with character-oriented protocols to ensure a minimum FIFO latency at the expense of performance.

RFW—Receive FIFO Width

0 = Receive FIFO is 32 bits wide for maximum performance. Data is not normally written to receive buffers until at least 32 bits are received. This configuration is required for HDLC-type protocols and Ethernet, but it is recommended for high-performance transparent modes. In this mode, the receive FIFO is 32 bytes for the serial communication controller.

1 = Low-latency operation. The receive FIFO is 8 bits wide and the receive FIFO is a quarter its normal size (8 bytes). This allows data to be written to the data buffer when a character is received, instead of waiting to receive 32 bits. This configuration must be chosen for character-oriented protocols, such as UART. It can also be used for low-performance, low-latency, transparent operation, if preferred. However, it must not be used with HDLC, HDLC Bus, AppleTalk, or Ethernet because it will cause erratic behavior to occur.

TXSY—Transmitter Synchronized to the Receiver

This bit is specifically intended for X.21 applications where the transmitted data must begin an exact multiple of 8-bit periods after the received data arrives.

0 = No synchronization between receiver and transmitter (default).

1 = The transmit bitstream is synchronized to the receiver. Additionally, if RSYN = 1, then transmission in the totally transparent mode does not occur until the receiver has synchronized with the bitstream and the $\overline{\text{CTS}}$ signal is asserted to the serial communication controller. Assuming $\overline{\text{CTS}}$ is already asserted, transmission begins eight clocks after the receiver starts receiving data.

SYNL—Sync Length (Transparent Mode Only)

This field determines the operation of the SCC2 receiver that is configured for totally transparent operation. See **Section 16.9.4 Data Synchronization Register** for more information.

00 = The sync pattern in the DSR is not used. An external sync signal is used instead.
01 = 4-bit sync. The receiver synchronizes on a 4-bit sync pattern stored in the DSR. This character and additional syncs can be programmed to be stripped using the Sync character in the parameter RAM. The transmitter transmits the entire contents of the DSR prior to each frame.
10 = 8-bit sync. The receiver synchronizes on an 8-bit sync pattern stored in the DSR. The transmitter transmits the entire contents of the DSR prior to each frame.
11 = 16-bit sync. The receiver synchronizes on a 16-bit sync pattern stored in the DSR. The transmitter transmits the DSR prior to each frame.

RTSM—$\overline{\text{RTS}}$ Mode

0 = Send idles between frames as defined by the protocol and the TEND bit in the GSMR_L. $\overline{\text{RTS}}$ is negated between frames (default).
1 = Send flags/syncs between frames according to the protocol. $\overline{\text{RTS}}$ is always asserted whenever the serial communication controller is enabled.

Note: This bit can be changed on-the-fly.

RSYN—Receive Synchronization Timing (Transparent Mode Only)

0 = Normal operation.
1 = If CDS = 1, then the $\overline{\text{CD}}$ signal should be asserted on the second bit of the receive frame, rather than the first.

GSMR_L

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RES	EDGE		TCI	TSNC		RINV	TINV	TPL			TPP		TEND	TDCR	
RESET	0	0		0	0		0	0	0			0		0	0	
R/W	R/W	R/W		R/W	R/W		R/W	R/W	R/W			R/W		R/W	R/W	
ADDR	(IMMR & 0xFFF0000) + 0xA20															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RDCR		RENC		TENC				DIAG		ENR	ENT	MODE			
RESET	0		0		0				0		0	0	0			
R/W	R/W		R/W		R/W				R/W		R/W	R/W	R/W			
ADDR	(IMMR & 0xFFF0000) + 0xA22															

Bit 0—Reserved

This bit is reserved and should be set to 0.

EDGE—Clock Edge

This field determines the clock edge the DPLL uses to adjust the received sample point after a jitter occurs in the received signal. These bits are ignored in the UART protocol or the x1 mode of the RDCR field.

00 = Both the positive and negative edges are used for changing the sample point (default).
01 = Positive edge. Only the positive edge of the received signal is used for changing the sample point.
10 = Negative edge. Only the negative edge of the received signal is used for changing the sample point.
11 = No adjustment is made to the sample point.

TCI—Transmit Clock Invert

0 = Normal operation.
1 = Before it is used, the internal transmit clock (TCLK) is inverted by the serial communication controller. This allows the serial communication controller to clock data out one-half clock earlier on the rising edge of TCLK rather than on the falling edge. In this mode, the serial communication controller offers a minimum and maximum "rising clock edge to data" specification. Data output by the serial communication controller after the rising edge of an external transmit clock can be latched by the external receiver one clock cycle later on the next rising edge of the same transmit clock. This option is recommended for Ethernet, HDLC, or Transparent operation when the clock rates are higher than 8MHz to improve data setup time for the external receiver.

TSNC—Transmit Sense

This bit indicates the amount of time the internal sense signal stays active after the last transition on the RXD2 pin, thus indicating that the line is free. For instance, TSNC can be used in the AppleTalk protocol to avoid a spurious $\overline{\text{CTS}}$-changed interrupt that would otherwise occur during the frame sync sequence preceding the opening flags.

If the RDCR field is configured to 1× mode, the delay is the greater of the two numbers listed. If RDCR is configured to 8×, 16×, or 32× mode, the delay is the lesser of the two numbers listed.

00 = Infinite—Carrier sense is always active (default).
01 = 14- or 6.5-bit times as determined by the RDCR field.
10 = 4- or 1.5-bit times as determined by the RDCR field (normally for AppleTalk).
11 = 3- or 1-bit times as determined by the RDCR field.

RINV—DPLL Receive Input Invert Data

This bit must be zero in HDLC bus mode.

0 = Do not invert.
1 = Invert the data before sending it to the on-chip DPLL for reception. This setting is used to produce FM1 from FM0 and NRZI space from NRZI mark. It can also be used in regular NRZ mode to invert the datastream.

TINV—DPLL Transmit Input Invert Data

This bit must be zero in HDLC bus mode.

0 = Do not invert.
1 = Invert the data before sending it to the on-chip DPLL for transmission. This setting is used to produce FM1 from FM0 and NRZI space from NRZI mark. It can also be used in regular NRZ mode to invert the datastream.

Note: In T1 applications, setting the TINV and TEND bits creates a continuously inverted HDLC datastream.

TPL—Transmit Preamble Length

This field determines the length of the preamble configured by the TPP field.

000 = No preamble (default).
001 = 8 bits (1 byte).
010 = 16 bits (2 bytes).
011 = 32 bits (4 bytes).
100 = 48 bits (6 bytes). Select this setting for Ethernet operation.
101 = 64 bits (8 bytes).
110 = 128 bits (16 bytes).
111 = Reserved.

TPP—Transmit Preamble Pattern

This field determines what, if any, bit pattern should precede the beginning of each transmit frame. The preamble pattern is sent prior to the first flag/sync of the frame. TPP is ignored if the serial communication controller is programmed to UART mode. The length of the preamble is programmed in the TPL field and the preamble pattern is typically transmitted to a receiving station that uses a DPLL for clock recovery. The receiving DPLL uses the regular pattern of the preamble to help it lock onto the received signal in a short, predictable time period.

00 = All zeros.
01 = Repetitive 10s. Select this setting for Ethernet operation.
10 = Repetitive 01s.
11 = All ones. Select this setting for LocalTalk operation.

TEND—Transmitter Frame Ending

This bit is specifically intended for NMSI transmitter encoding of the DPLL. This bit determines whether the TXD2 signal should idle in a high state or in an encoded ones state (high or low). It can, however, be used with other encodings besides NMSI.

0 = Default operation. The TXD2 signal is only encoded when data is transmitted, including the preamble and opening and closing flags/syncs. When no data is available to transmit, the signal is driven high.
1 = The TXD2 signal is always encoded, even when idles are transmitted.

TDCR—Transmit Divide Clock Rate

This field determines the divider rate of the transmitter. If the DPLL is not used, the 1× value should be chosen, except in asynchronous UART mode where 8×, 16×, or 32× must be chosen. You should program TDCR to equal RDCR in most applications. If the DPLL is used in the application, the selection of TDCR depends on the encoding. NRZI usually requires 1×, whereas FM0/FM1, Manchester, and Differential Manchester allow 8×, 16×, or 32×. The 8× option allows highest speed, whereas the 32× option provides the greatest resolution.

TDCR is usually equal to RDCR so that the same clock frequency source can control both the transmitter and receiver.

00 = 1× clock mode. Only NRZ or NRZI encodings are allowed.
01 = 8× clock mode.
10 = 16× clock mode. Normally chosen for UART and AppleTalk.
11 = 32× clock mode.

RDCR—Receive DPLL Clock Rate

This field determines the divider rate of the receive DPLL. If the DPLL is not used, the 1× value should be chosen, except in asynchronous UART mode where 8×, 16×, or 32× must be chosen. You should program this field to equal TDCR in most applications.

If the DPLL is used in the application, the selection of RDCR depends on the encoding. NRZI usually requires 1×, whereas FM0, FM1, Manchester, and Differential Manchester allow 8×, 16×, or 32×. The 8× option allows highest speed, whereas the 32× option provides the greatest resolution.

00 = 1× clock mode. Only NRZ or NRZI decodings are allowed.
01 = 8× clock mode.
10 = 16× clock mode. Normally chosen for UART and AppleTalk.
11 = 32× clock mode.

RENC—Receiver Decoding Method

000 = NRZ (default setting if DPLL is not used).
001 = NRZI Mark (set RINV also for NRZI space).
010 = FM0 (set RINV also for FM1).
011 = Reserved.
100 = Manchester.
101 = Reserved.
110 = Differential Manchester (Differential Bi-phase-L).
111 = Reserved.

Note: Select NRZ if the DPLL is not used. You should program this bit to equal TENC in most applications. However, do not use this internal DPLL for Ethernet mode.

TENC—Transmitter Encoding Method

000 = NRZ (default setting if DPLL is not used).
001 = NRZI Mark (set TINV also for NRZI space).
010 = FM0 (set TINV also for FM1).
011 = Reserved.
100 = Manchester.
101 = Reserved.
110 = Differential Manchester (Differential Bi-phase-L).
111 = Reserved.

Note: Select NRZ if the DPLL is not used. You should program this bit to equal RENC in most applications. However, do not use this internal DPLL for Ethernet mode.

DIAG—Diagnostic Mode

The data received enters the RXD2 pin and the transmit data is shifted out through the TXD2 pin. The serial communication controller uses the modem signals to automatically enable or disable transmission and reception. These timings are shown in **Section 16.9.10 Controlling SCC2 Timing**.

In local loopback mode, the transmitter output is internally connected to the receiver input, while the receiver and the transmitter operate normally. The value on the RXD2 pin is ignored. Data can be programmed to appear on the TXD2 or the port A register can be programmed to make the TXD2 pin high. See **Section 16.14 The Parallel I/O Ports** for more information. The $\overline{\text{RTS}}$ pin can also be programmed to be disabled in the appropriate parallel I/O register. In the SDMA field of the SIMODE register, the L1TXDA and L1RQA signals can be programmed to be either asserted normally or to remain inactive. See **Section 16.7.5.2 Serial Interface Mode Register** for more information.

When using local loopback mode, the clock source for the transmitter and receiver must be the same. Thus, the same baud rate generator or external CLKx pin can be used for both the transmitter and receiver. Separate CLKx pins can be used with the transmitter and receiver as long as the CLKx pins are connected to the same external clock signal source.

In automatic echo mode, the channel automatically retransmits the received data on a bit-by-bit basis using whatever receive clock is provided. The receiver operates normally and receives data if $\overline{CD}$ is asserted. The transmitter simply transmits the received data. In this mode, the $\overline{CTS}$ signal is ignored. The echo function can be implemented in the software by receiving buffers from a serial communication controller, linking them to TX buffer descriptors, and then transmitting them back out of that serial communication controller.

In loopback/echo mode, loopback and echo operation occur simultaneously. The $\overline{CD}$ and $\overline{CTS}$ pins are ignored. Refer to the loopback bit description for clocking requirements.

00 = Normal operation. The $\overline{CTS}$ and $\overline{CD}$ signals are under automatic control.
01 = Local loopback mode.
10 = Automatic echo mode.
11 = Loopback and echo mode.

Note: If you prefer external loopback, the DIAG field should be selected for normal operation and an external connection should be made between the TXD2 and RXD2 pins. Clocks can be generated internally using a baud rate generator or generated externally. You can physically connect the appropriate control signals or the port C register can be used to cause the $\overline{CD}$ and $\overline{CTS}$ pins to be permanently asserted to the serial communication controller.

ENR—Enable Receive

This bit enables the receiver hardware state machine for the serial communication controller. When ENR is cleared, the receiver is disabled and any data in the receive FIFO is lost. If ENR is cleared during reception, the receiver aborts the current character. ENR may be set or cleared, regardless of whether the serial clocks are present. Refer to **Section 16.9.14 Disabling the SCC2 On-the-Fly** for a description of how to disable and reenable the serial communication controller.

Note: The serial communication controller that controls reception provides other tools besides the ENR bit. Some of these tools include the **ENTER HUNT MODE** command, **CLOSE RX BD** command, and the E bit of the RX buffer descriptor.

ENT—Enable Transmit

This bit enables the transmitter hardware state machine for this serial communication controller. When ENT is cleared, the transmitter is disabled. If ENT is cleared during transmission, the transmitter aborts the current character and the TXD2 pin returns to the idle state. Data already in the transmit shift register is not transmitted. ENT can be set or cleared, regardless of whether the serial clocks are present. Refer to **Section 16.9.14 Disabling the SCC2 On-the-Fly** for a description of how to properly disable and reenable the serial communication controller.

Note: The serial communication controller that controls transmission provides other tools besides the ENT bit. These tools include the **STOP TRANSMIT** command, **GRACEFUL STOP TRANSMIT** command, **RESTART TRANSMIT** command, the freeze option in UART mode, $\overline{\text{CTS}}$ flow control option in UART mode, and the R bit of the TX buffer descriptor.

MODE—Channel Protocol Mode

0000 = HDLC.
0001 = Reserved.
0010 = AppleTalk/LocalTalk.
0011 = SS7. Reserved for RAM microcode.
0100 = UART.
0101 = Profibus. Reserved for RAM microcode.
0110 = ASYNC HDLC/IrDA.
0111 = V.14. Reserved for RAM microcode.
1000 = Reserved.
1001 = DDCMP. Reserved for RAM microcode.
1010 = Reserved.
1011 = Reserved.
1100 = Ethernet.
11xx = Reserved.

16.9.3 Protocol-Specific Mode Register

The functionality of the serial communication controller varies according to the protocol selected by the MODE field in the GSMR_L. The serial communication controller has an additional 16-bit, memory-mapped, read/write protocol-specific mode register (PSMR) that configures it specifically for a chosen mode. Since each protocol has specific requirements, the PSMR bits are different for each implementation. A detailed description of each of the PSMR bits is contained within each specific SCC2 protocol.

16.9.4 Data Synchronization Register

The serial communication controller has a 16-bit, memory-mapped, read/write data synchronization register (DSR) that specifies the pattern used in the frame synchronization procedure of the synchronous protocols. In the UART protocol, it is used to configure fractional stop bit transmission. In the Transparent protocol, it should be programmed with the preferred Sync pattern. In the Ethernet protocol, it should be programmed with 0xD555. At reset, it defaults to 0x7E7E (two HDLC flags), so it does not need to be written for HDLC mode. When the DSR is used to send out syncs (such as in transparent mode), the contents of the DSR are always transmitted LSB first.

DSR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	SYN2								SYN1							
RESET	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1	0
R/W	R/W								R/W							
ADDR	(IMMR & 0xFFF0000) + 0xA2E															

16.9.5 Transmit-on-Demand Register

If no frame is currently being transmitted by the serial communication controller, the RISC microcontroller periodically polls the R bit of the next TX buffer descriptor to see if you have requested a new frame/buffer to be transmitted. This polling algorithm depends on the serial communication controller configuration, but occurs every 8 to 32 serial transmit clocks. You can, however, request that the RISC microcontroller begin processing the new frame/buffer immediately, without waiting until the normal polling time. To obtain immediate processing, the TOD bit in the transmit-on-demand register (TODR) is set after the R bit is set in the TX buffer descriptor.

This feature, which decreases the transmission latency of the transmit buffer/frame, is particularly useful in LAN-type protocols where maximum interframe GAP times are limited by the protocol specification. Since the transmit-on-demand feature gives high priority to the specified TX buffer descriptor, it can conceivably affect the servicing of the receive FIFO. Therefore, it is recommended that you only use the transmit-on-demand feature when a high-priority TX buffer descriptor has been prepared and if a sufficient amount of time has passed since the serial communication controller was transmitted.

The TOD bit does not need to be set if a new TX buffer descriptor is added to the circular queue or if other TX buffer descriptors in that queue have not finished transmitting. Once they are finished, however, the new TX buffer descriptors are processed. The first bit of the frame will typically be clocked out 5-6 bit times after the TOD bit has been set to 1.

TODR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	TOD	RESERVED														
RESET	0	0														
R/W	R/W	R/W														
ADDR	0xA2C															

TOD—Transmit on Demand

0 = Normal operation.

1 = The RISC microcontroller gives high priority to the current TX buffer descriptor and does not wait the normal polling time to check that the TX buffer descriptor's R bit has been set. Instead, it begins transmitting the frame. This bit is automatically cleared after one serial clock.

Bits 1–15—Reserved

These bits are reserved and should be set to 0.

16.9.6 SCC2 Buffer Descriptor Operation

Data associated with the serial communication controller channel transmitter and receiver is stored in buffers and each buffer is referenced by a buffer descriptor that can be located anywhere in internal memory. The MPC823 internal memory has enough space for 224 buffer descriptors (BDs) to be shared between the universal serial bus, serial communication controller, serial managment controllers, serial peripheral interface, and I^2C controller. However, you must define how the buffer descriptors will be allocated to the serial channel's transmitter or receiver. You can select 100 buffer descriptors for the SCC2 receiver or 20 buffer descriptors for the transmitter.

The buffer descriptor table forms a circular queue with a programmable length. You can program the start address of each channel buffer descriptor table in the internal memory. In addition, you can allocate the parameter area of an unused channel to the other used channels as buffer descriptor tables or actual buffers. See Figure 16-63 for details.

Note: The communication processor module sets all the status bits in these buffer descriptors. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

The format of the buffer descriptors is the same for each serial communication controller mode of operation and for transmit and receive. The first word in each buffer descriptor contains a status and control word that determines the buffer descriptor's table length. Only this first field, which contains the status and control bits, differs for each protocol. The second word determines the data length referenced to this buffer descriptor and the last two words in the buffer descriptor contain a 32-bit address pointer that points to the actual buffer in memory.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	R/E	—	W	I	STATUS AND CONTROL											
OFFSET + 2	DATA LENGTH															
OFFSET + 4	HIGH-ORDER DATA BUFFER POINTER															
OFFSET + 6	LOW-ORDER DATA BUFFER POINTER															

R/E—Ready/Empty

Ready (Transmitter):

0 = The data buffer associated with this buffer descriptor is not ready to be transmitted. You are free to manipulate this buffer descriptor or its associated data buffer. The communication processor module clears this bit after the buffer is transmitted or after an error condition is encountered.

1 = The data buffer, which you must prepare for transmission, has not been transmitted yet or is currently being transmitted. You cannot write any fields of this buffer descriptor once this bit is set.

Empty (Receiver):

0 = The data buffer associated with this RX buffer descriptor has been filled with data or reception has been aborted because of an error condition. The core is free to examine or write to any fields of this RX buffer descriptor. The communication processor module does not use this buffer descriptor again as long as the E bit is zero.

1 = The data buffer associated with this buffer descriptor is empty or is currently receiving data. This RX buffer descriptor and its associated receive buffer are owned by the communication processor module. Once the E bit is set, the core should not write any fields of this RX buffer descriptor.

W—Wrap

0 = This is not the last buffer descriptor in the buffer descriptor ring.

1 = This is the last buffer descriptor in the buffer descriptor ring. After this buffer has been used, the communication processor module will transmit (receive) data from the first buffer descriptor that TBASE (RBASE) points to in the table. The number of TX buffer descriptors in the ring are programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is serviced.
1 = The specific RX or TX bit in the protocol event register is set when this buffer is serviced by the communication processor module. These bits can cause an interrupt when enabled by the mask register.

For frame-oriented protocols, a message can reside in as many buffers as necessary. Each buffer has a maximum length of (64K–1) bytes. The communication processor module does not assume that all buffers of a single frame are currently linked to the buffer descriptor table. It does assume, however, that the unlinked buffers are provided by the core in time to be transmitted or received. When this does not occur, an error condition is reported by the communication processor module. An underrun error is reported when data is transmitted and a busy error is reported when data is received.

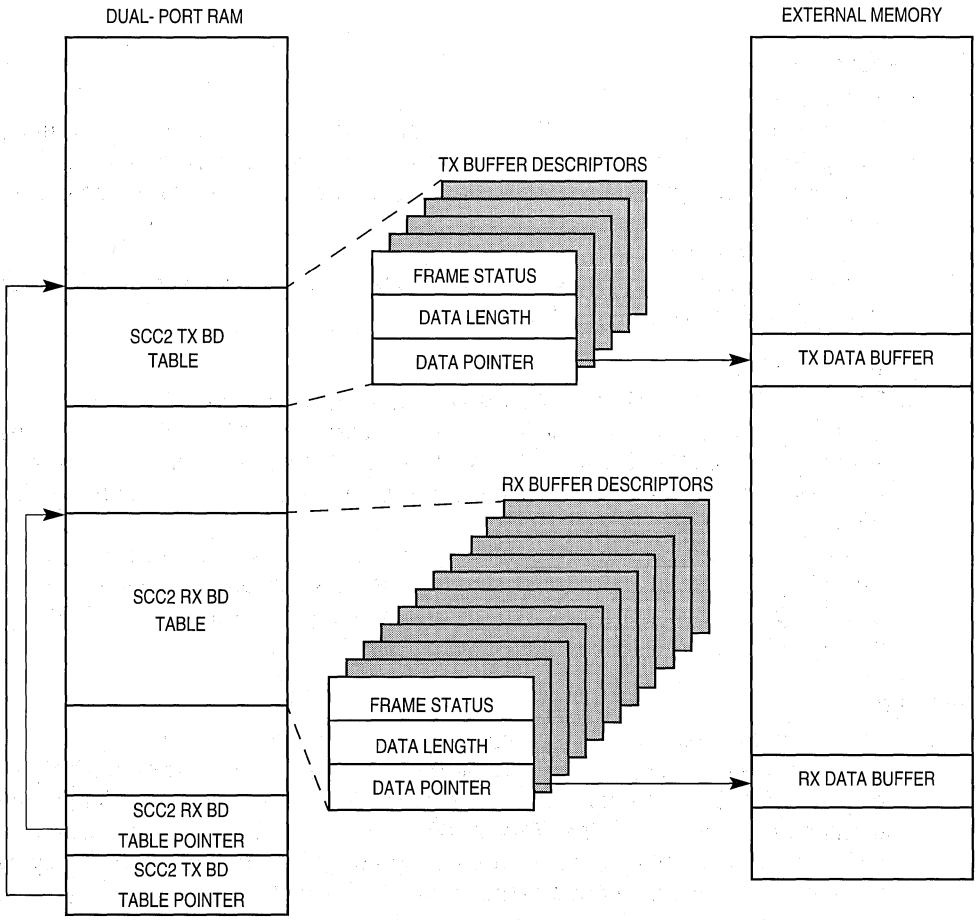


Figure 16-63. SCC2 Memory Structure

All protocols can have their buffer descriptors point to data buffers that are located in the internal dual-port RAM. However, because the internal RAM is being used for buffer descriptors, it is customary for the data buffers to be located in external RAM, especially if the data buffers are large. In most instances, the internal U-Bus is used to transfer data to the data buffer.

The communication processor module processes the TX buffer descriptors in a straightforward manner. Once the transmit side of a serial communication controller is enabled, it starts with the first buffer descriptor in that SCC2 transmit table. Once the communication processor module detects that the R bit is set in the TX buffer descriptor, it starts processing the buffer. The communication processor module detects that the buffer descriptor is ready when it polls the R bit or when you try to write to the TODR. Once the data from the buffer descriptor has been placed in the transmit FIFO, the communication processor module moves on to the next buffer descriptor, again waiting for that buffer descriptor R bit to be set. Thus, the communication processor module does no look-ahead buffer descriptor processing, nor does it skip over buffer descriptors that are not ready.

When the communication processor module sees the W bit set in a buffer descriptor, it goes back to the beginning of the buffer descriptor table after processing of the buffer descriptor is complete. After using a buffer descriptor, the communication processor module normally sets the R bit to 0, thus, the communication processor module does not use a buffer descriptor twice until the buffer descriptor has been confirmed by the core. The one exception to this rule is that the MPC823 supports an option for repeated transmission called continuous mode, whereby the R bit stays set to 1. This is available in some protocols.

The communication processor module uses the RX buffer descriptors in a similar fashion. Once the receive side of a serial communication controller is enabled, it starts with the first buffer descriptor in that SCC2 RX buffer descriptor table. Once data arrives from the serial line into the serial communication controller, the communication processor module performs certain required protocol processing on the data and moves the resultant data to the data buffer pointed to by the first buffer descriptor. Use of a buffer descriptor is complete when there is no more room left in the buffer or when certain events occur, such as detection of an error or an end-of-frame. Whatever the reason, the buffer is then closed and additional data is stored using the next buffer descriptor.

Whenever the communication processor module needs to begin using a buffer descriptor because new data is arriving, it checks the E bit of that buffer descriptor. If the current buffer descriptor is not empty, it reports a busy error. However, it does not move from the current buffer descriptor until it is empty. When the communication processor module discovers the W bit set in a buffer descriptor, it goes back to the beginning of the buffer descriptor table after processing is complete and after using a buffer descriptor, the communication processor module sets the E bit to 0 and never uses a buffer descriptor twice until it has been processed by the core. The one exception to this rule is that the MPC823 supports an option for repeated reception called continuous mode, whereby the E bit stays set to 1. This is available in some protocols.

16.9.7 SCC2 Parameter RAM Memory Map

The serial communication controller parameter RAM area begins at an offset from the SCC2 base area. The protocol-specific portions of the SCC2 parameter RAM are discussed in the specific protocol descriptions and the part that is common to all serial communication controller protocols is shown in Table 16-24.

You need to initialize certain parameter RAM values before the serial communication controller can be enabled. Other values are initialized or written by the communication processor module. Once initialized, most parameter RAM values do not need to be accessed in your software because most of the activity is centered around the transmit and receive buffer descriptors, not the parameter RAM. However, if you do access the parameter RAM, the following regulations should be noted:

- The parameter RAM can be read at any time.
- The parameter time values related to the SCC2 transmitter can only be written whenever the transmitter is disabled after a **STOP TRANSMIT** command and before a **RESTART TRANSMIT** command or after the buffer/frame finishes transmitting as a result of the execution of a **GRACEFUL STOP TRANSMIT** command and before a **RESTART TRANSMIT** command is executed.
- The parameter RAM values related to the SCC2 receiver can only be written when the receiver is disabled.

Refer to **Section 16.9.14 Disabling the SCC2 On-the-Fly** for more information.

Table 16-24. SCC2 Parameter RAM Memory Map For All Protocols

ADDRESS	NAME	WIDTH	DESCRIPTION
SCC2 Base + 00	**RBASE**	Half-word	RX BD Base Address
SCC2 Base + 02	**TBASE**	Half-word	TX BD Base Address
SCC2 Base + 04	**RFCR**	Byte	RX Function Code
SCC2 Base + 05	**TFCR**	Byte	TX Function Code
SCC2 Base + 06	**MRBLR**	Half-word	Maximum Receive Buffer Length
SCC2 Base + 08	RSTATE	Word	RX Internal State
SCC2 Base + 0C	RPTR	Word	RX Internal Data Pointer
SCC2 Base + 10	RBPTR	Half-word	RX BD Pointer
SCC2 Base + 12	RCNT	Half-word	RX Internal Byte Count
SCC2 Base + 14	RTMP	Word	RX Temp
SCC2 Base + 18	TSTATE	Word	TX Internal State
SCC2 Base + 1C	TPTR	Word	TX Internal Data Pointer
SCC2 Base + 20	TBPTR	Half-word	TX BD Pointer
SCC2 Base + 22	TCNT	Half-word	TX Internal Byte Count
SCC2 Base + 24	TTMP	Word	TX Temp
SCC2 Base + 28	RCRC	Word	Temp Receive CRC
SCC2 Base + 2C	TCRC	Word	Temp Transmit CRC
SCC2 Base + 30			First Word of Protocol-Specific Area
SCC2 Base + xx			Last Word of Protocol-Specific Area

NOTE: You are only responsible for initializing the items in bold.
SCC2 Base = (IMMR & 0xFFFF0000) + 0x3D00.

- RBASE and TBASE—These entries are where the dual-port RAM starts the SCC2 receive and transmit buffer descriptors. They provide a great deal of flexibility in how buffer descriptors for the serial communication controller are partitioned. By selecting RBASE and TBASE entries for the serial communication controller and by setting the W bit in the last buffer descriptor in each buffer descriptor list, you can select how many buffer descriptors to allocate for the transmit and receive side of the serial communication controller. However, you must initialize these entries before enabling the corresponding channel.

Note: RBASE and TBASE should contain a value that is divisible by eight.

- RFCR and TFCR—There are two function code registers for the SCC2 channel—one for receive data buffers and one for transmit data buffers. The function code entry contains the value that you want to appear on the AT pins when the associated SDMA channel accesses memory. It also controls the byte-ordering convention for transfers.

RFCR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED			BO		AT		
RESET	0			0		0		
R/W	R/W			R/W		R/W		
ADDR	SCC2 BASE + 0x00							

Bits 0–2—Reserved

These bits are reserved and should be set to 0.

BO—Byte Ordering

You should set these bits to select the required byte ordering of the data buffer. If this field is modified on-the-fly, it takes effect at the beginning of the next frame or the next buffer descriptor.

00 = The DEC/Intel convention is used for byte ordering (swapped operation) and is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation) is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT—Address Type

These bits contain the function code value used during the SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA type.

TFCR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED			BO		AT1–AT3		
RESET	0			0		0		
R/W	R/W			R/W		R/W		
ADDR	SCC2 BASE + 0x02							

Bits 0–2—Reserved

These bits are reserved and should be set to 0.

BO—Byte Ordering

You should set these bits to select the required byte ordering of the data buffer. If this field is modified on-the-fly, it takes effect at the beginning of the next frame or the next buffer descriptor.

00 = The DEC/Intel convention is used for byte ordering (swapped operation) and is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation) is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT—Address Type

These bits contain the function code value used during this SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA type.

- MRBLR—The serial communication controller has one maximum receive buffer length register to define the receive buffer length. The MRBLR defines the maximum number of bytes that the MPC823 writes to a receive buffer on the serial communication controller before it moves on to the next buffer. The MPC823 can write fewer bytes to the buffer than MRBLR if a condition, such as an error or end-of-frame occurs, but it never writes more bytes than the MRBLR value. It follows then, that buffers you supply should always be at least as long as the MRBLR. The transmit buffers for the serial communication controller are not affected in any way by the value programmed into the MRBLR. Transmit buffers can be individually chosen to have varying lengths. The number of bytes to be transmitted is chosen by programming the DATA LENGTH field in the TX buffer descriptor.

Note: The MRBLR is not intended to be dynamically changed while the serial communication controller is operating. However, if it is modified in a single bus cycle with one 16-bit move, then a dynamic change in receive buffer length can be successfully achieved. This occurs when the communication processor module moves control to the next RX buffer descriptor in the table. Thus, a change to MRBLR does not have an immediate effect. To guarantee the exact RX buffer descriptor on which the change occurs, you should only change the MRBLR while the SCC2 receiver is disabled. The value of MRBLR should be greater than zero for all modes. For Ethernet and HDLC mode, the MRBLR should be evenly divisible by 4. In transparent mode, the MRBLR should also be divisible by 4, unless the RFW bit of the GSMR_H is set to 8 bits.

- RBPTR—The receiver buffer descriptor pointer for each SCC2 channel points to the next buffer descriptor the receiver transfers data to when it is in idle state or to the current buffer descriptor during frame processing. After a reset or when the end of the buffer descriptor table is reached, the communication processor module initializes this pointer to the value programmed in the RBASE register. Although you will not usually need to write the RBPTR in most applications, you can modify it when the receiver is disabled or when you are sure no receive buffer is currently being used.
- TBPTR—The transmitter buffer descriptor pointer for each SCC2 channel points to the next buffer descriptor the transmitter transfers data from when it is in idle state or to the current buffer descriptor during frame transmission. After a reset or when the end of the buffer descriptor table is reached, the communication processor module initializes this pointer to the value programmed in the TBASE register. Although you will not usually need to write TBPTR in most applications, you can modify it when the receiver is disabled or when you are sure no receive buffer is currently being used.
- Other General Parameters—You do not need to access these parameters during normal operation. They are only listed because they provide helpful information for experienced users and for debugging purposes. Additional parameters are listed in Table 16-24. RPTR and TPTR are updated by the SDMA channels to show the next address in the buffer to be accessed. TCNT is a down-count value that is initialized with the TX buffer descriptors data length and decremented with every byte read by the SDMA channels. RCNT is a down-count value that is initialized with the MRBLR value and decremented with every byte written by the SDMA channels. RSTATE, TSTATE, RTMP, TTMP, and reserved areas should only be used by the RISC microcontroller.

Note: To extract data from a partially full receive buffer, issue the **CLOSE RX BD** command.

16.9.8 Handling Interrupts In the SCC2

Interrupt handling for the SCC2 channel is configured on a global basis in the CPM interrupt pending register, CPM interrupt mask register, and CPM in-service register. In each of these registers, a bit is used to mask, enable, or report the presence of an interrupt from SCC2. The interrupt priority between the serial communication controller is programmable in the CPM interrupt configuration register. There is an event register within each protocol of the serial communication controller that allows interrupt handling.

A number of events can cause the serial communication controller to interrupt the processor and these events differ slightly depending on the protocol you have selected. These events are handled independently between each channel by the SCC2 event and mask registers. Events that can cause interrupts because of the $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ modem lines are described in **Section 16.14.8 Port C Pin Functionality**.

16.9.8.1 SCC2 EVENT REGISTER. The 16-bit memory-mapped SCC2 event (SCCE) register is used to report events recognized by the serial communication controller. When an event is recognized, the serial communication controller sets the corresponding bit in the SCCE, regardless of the corresponding mask bit. A bit is cleared by writing a 1 (writing a zero has no effect) and more than one bit can be cleared at a time. This register is cleared at reset and can be read at any time. Since each protocol has specific requirements, the protocol-specific mode register (PSMR) bits are different for each implementation. A detailed description of each of the PSMR bits is contained within each specific protocol.

16.9.8.2 SCC2 MASK REGISTER. The 16-bit, read/write SCC2 mask (SCCM) register allows you to enable or disable interrupt generation using the communication processor module for specific events in each SCC2 channel. An interrupt is only generated if the SCC2 interrupts in this channel are enabled in the CPM interrupt mask register (CIMR).

If a bit in the SCCM register is zero, the communication processor module does not proceed with its usual interrupt handling whenever that event occurs. Anytime a bit in the SCCM register is set, a 1 in the corresponding bit in the SCCE register sets the SCC2 bit in the CPM interrupt pending register (CIPR), which is described in **Section 16.15 The CPM Interrupt Controller**. The bit format of the SCCM register is identical to that of the SCCE. Since each protocol has specific requirements, the SCCM bits are different for each implementation. A detailed description of each of the SCCM bits is contained within each specific protocol.

16.9.8.3 SCC2 STATUS REGISTER. The 8-bit, read/write SCC2 status (SCCS) register allows you to monitor real-time status conditions on the RXD2 signal. It does not show the real-time status of the $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ pins. Their real-time status is available in the port C parallel I/O. Since each protocol has specific requirements, the SCCS bits are different for each implementation. A detailed description of each of the SCCS bits is contained within each specific protocol.

16.9.9 Initializing Serial Communication Controller

The serial communication controller requires that a number of registers and parameters be configured after a power-on reset. Regardless of the protocol you are using, follow these steps to initialize the serial communication controller:

1. Write the parallel I/O ports to configure and connect the I/O pins to the serial communication controller.
2. The RAID field of the SDCR should be initialized with the appropriate arbitration ID.
3. Write the port C registers to configure the $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ pins to be in parallel I/O with interrupt capability or to be direct connections to the serial communication controller (if modem support is needed).
4. If the time-slot assigner is used, the serial interface must be configured. If the serial communication controller is used in NMSI mode, the SICR must still be initialized.
5. Write the GSMR_H and GSMR_L, but do not write the ENT or ENR bits yet.
6. Write the PSMR.
7. Write the DSR.
8. Initialize the required values for the serial communication controller in its parameter RAM.
9. Clear out any current events in the SCCE register (optional).
10. Write the SCCM register to enable the interrupts in the SCCE register.
11. Write the CICR to configure the SCC2 interrupt priority.
12. Clear out any current interrupts in the CIPR (optional).
13. Write the CIMR to enable interrupts to the CPM interrupt controller.
14. Set the ENT and ENR bits in the GSMR_L.

The buffer descriptors can have their R or E bits set at any time. Notice that the CPCR does not need to be accessed after a power-on reset. The serial communication controller should be disabled and reenabled after any dynamic change in its parallel I/O ports or serial channel physical interface configuration. A full reset using the RST bit in the CPCR is a comprehensive reset that can also be implemented.

Follow these steps to handle an interrupt in the serial communication controller:

1. Once an interrupt occurs, read the SCCE register to find out which source has caused the interrupt. The SCCE bits that are going to be "handled" in this interrupt handler would normally be cleared at this time by writing ones to them.
2. Process the TX buffer descriptors to reuse them if the TX or TXE bit was set in the SCCE register. If the transmit speed is fast or the interrupt delay is long, more than one transmit buffer may have been sent by the serial communication controller. Thus, it is important to check more than just one TX buffer descriptor during interrupt handling. One common practice is to process all TX buffer descriptors in the interrupt handler until one is found with its R bit set.

3. Extract data from the RX buffer descriptor if the RX, RXB, or RXF bit is set in the SCCE register. If the receive speed is fast or the interrupt delay is long, more than one receive buffer may have been received by the serial communication controller. Thus, it is important to check more than just one RX buffer descriptor during interrupt handling. One common practice is to process all RX buffer descriptors in the interrupt handler until one is found with its E bit set.
4. Reset the status bit in the buffer descriptor's control and status field that is associated with the interrupt. These bits do not set reset after each I/O operation.
5. Clear the SCC2 bit in the CISR.
6. Execute the **rfi** instruction.

16.9.10 Controlling SCC2 Timing

When the DIAG field of the GSMR_L are programmed to normal operation, the $\overline{CD}$ and $\overline{CTS}$ signals are automatically controlled by the serial communication controller. It is assumed that the TCI bit of the GSMR_L is cleared, which implies a normal transmit clock operation.

16.9.10.1 SYNCHRONOUS PROTOCOLS. In synchronous protocols, the $\overline{RTS}$ pin is asserted when the serial communication controller data is loaded into the transmit FIFO and a falling transmit clock occurs. At this point, the serial communication controller starts transmitting data once the appropriate conditions occur on the $\overline{CTS}$ pin. In all cases, the first bit of data is the start of the opening flag, sync pattern, or preamble.

Figure 16-64 illustrates that the delay between the $\overline{RTS}$ pin and data is 0 bit times, regardless of how the CTSS bit is set in the GSMR_H. This operation assumes that the $\overline{CTS}$ pin is already asserted to the serial communication controller or that the $\overline{CTS}$ pin is reprogrammed to be a parallel I/O line, in which case the $\overline{CTS}$ signal to the serial communication controller is always asserted. The $\overline{RTS}$ pin is negated one clock after the last bit in the frame.

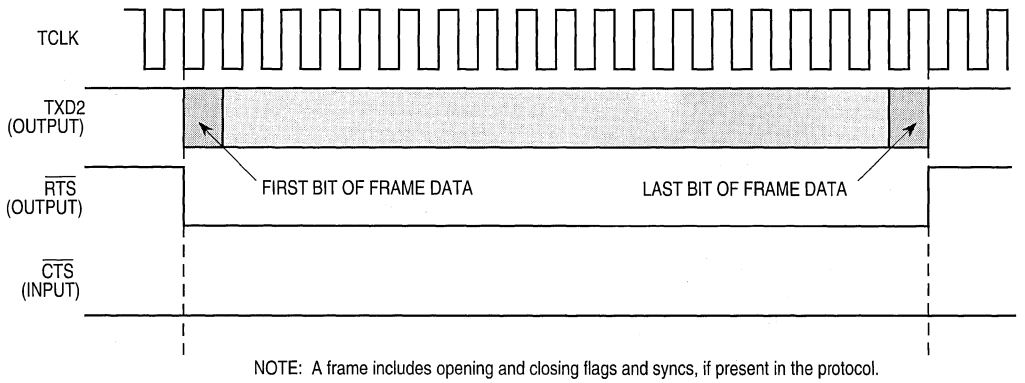


Figure 16-64. $\overline{RTS}$ Output Delays Asserted for Synchronous Protocols

If the $\overline{CTS}$ pin is not already asserted when the $\overline{RTS}$ pin is asserted, then the delays to the first bit of data depend on when $\overline{CTS}$ is asserted. Figure 16-65 illustrates that the delay between $\overline{CTS}$ and the data can be approximately 0.5- to 1-bit time or 0-bit times, depending on how the CTSS bit is set in the GSMR_H.

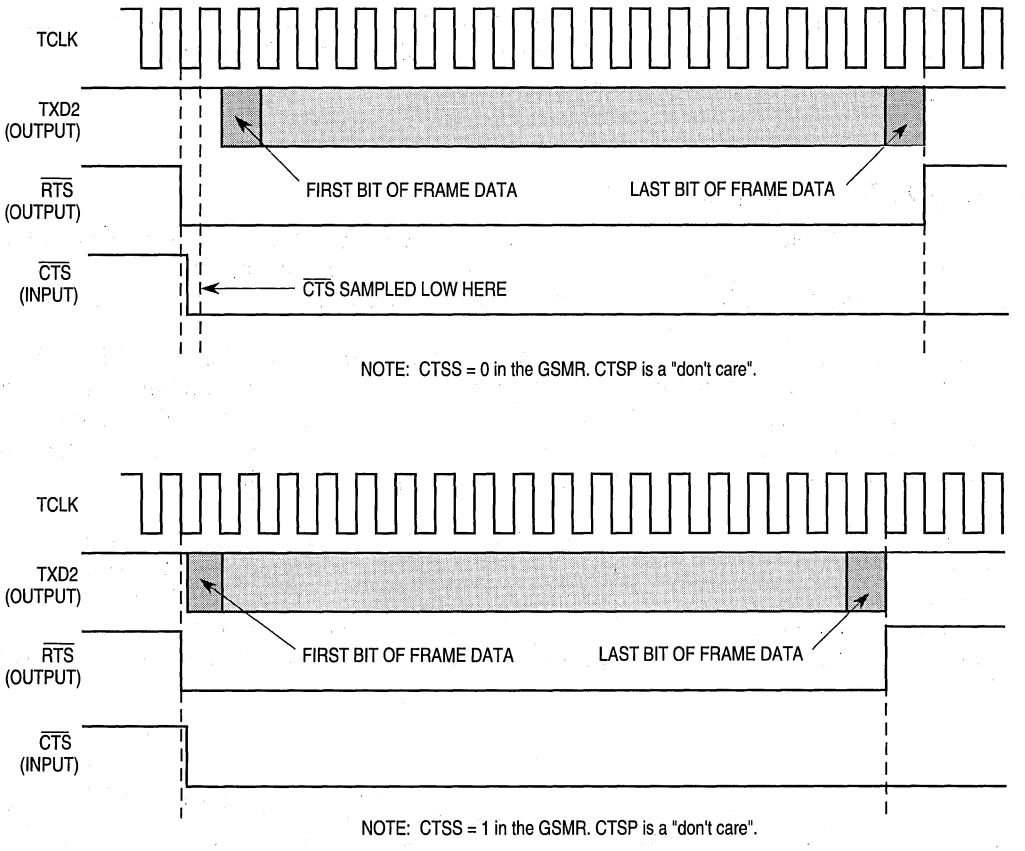


Figure 16-65. $\overline{CTS}$ Output Delays Asserted for Synchronous Protocols

If the $\overline{CTS}$ pin is programmed to envelope the data, it must remain asserted during frame transmission or a $\overline{CTS}$ lost error occurs. Negation of the $\overline{CTS}$ pin forces the $\overline{RTS}$ pin high and the transmit data to an idle state. If the CTSS bit in the GSMR_H is zero, the $\overline{CTS}$ pin must be sampled by the serial communication controller before a $\overline{CTS}$ lost is recognized. Otherwise, the negation of $\overline{CTS}$ immediately causes the $\overline{CTS}$ lost condition. Refer to Figure 16-66 for details.

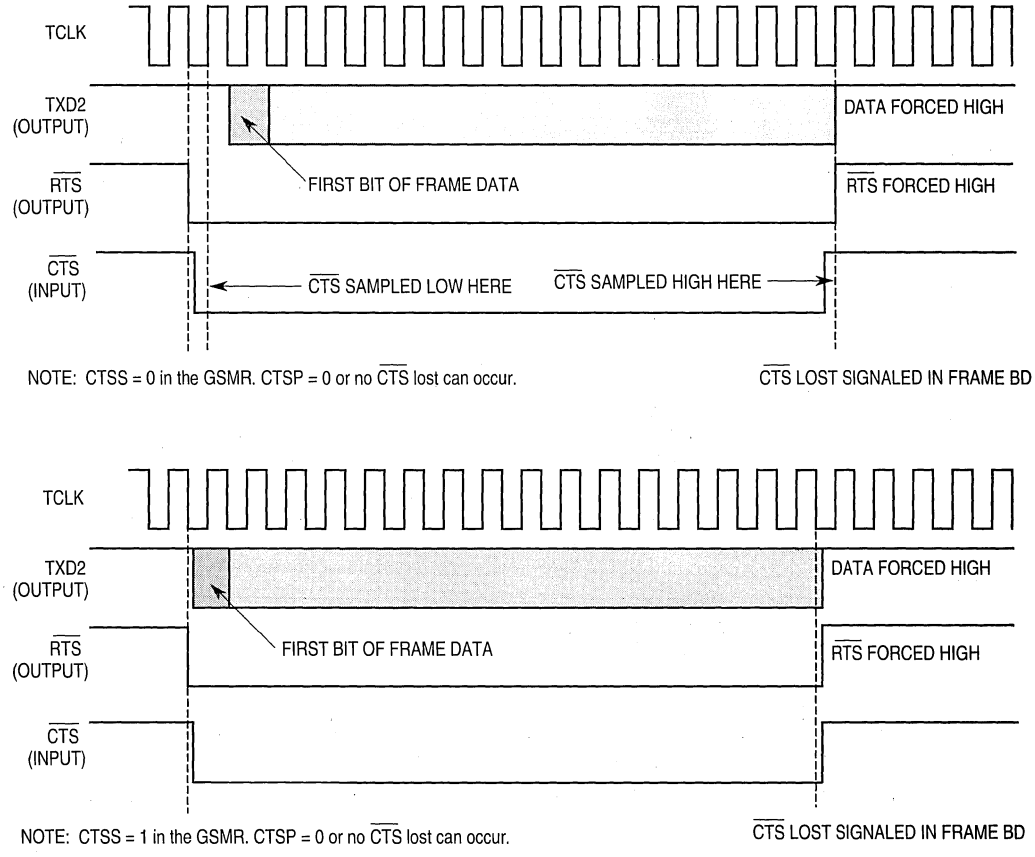
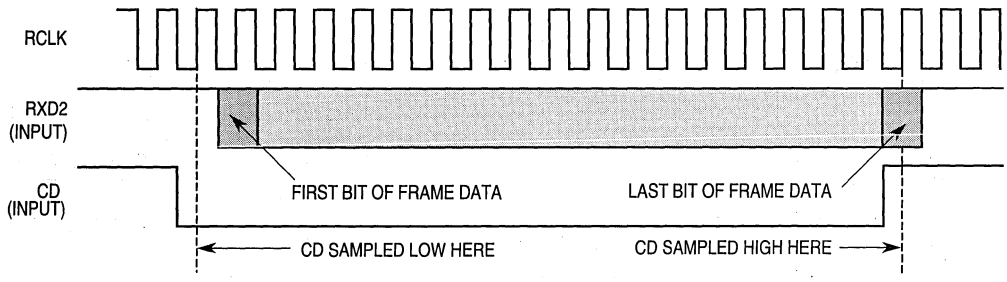


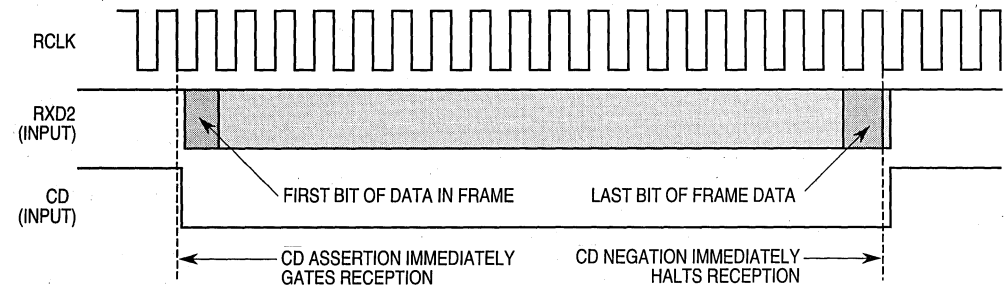
Figure 16-66. $\overline{CTS}$ Lost in Synchronous Protocols

Note: If the CTSS bit in the GSMR_H is set, all $\overline{CTS}$ transitions must occur while the transmit clock is low.

Reception delays are determined by the $\overline{CD}$ pin as illustrated in Figure 16-67. If the CDS bit in the GSMR_H is zero, then the $\overline{CD}$ pin is sampled on the rising receive clock edge prior to data being received. If the CDS bit in the GSMR_H is 1, then the $\overline{CD}$ pin transitions cause data to be immediately gated into the receiver.



NOTES:
1. CDS = 0 in the GSMR; CDP = 0.
2. If CD is negated prior to the last bit of the receive frame, CD lost is signaled in the frame buffer descriptor.
3. If CDP = 1, CD lost cannot occur and CD negation has no effect on reception.



NOTES:
1. CDS = 1 in the GSMR; CDP = 0.
2. If CD is negated prior to the last bit of the receive frame, CD lost is signaled in the frame buffer descriptor.
3. If CDP = 1, CD lost cannot occur and CD negation has no effect on reception.

Figure 16-67. Using $\overline{CD}$ to Control Synchronous Protocol Reception

If the $\overline{CD}$ pin is programmed to envelope the data, it must remain asserted during frame transmission or a $\overline{CD}$ lost error occurs. Negation of the $\overline{CD}$ pin terminates reception. If the CDS bit in the GSMR_H is zero, the $\overline{CD}$ pin must be sampled by the serial communication controller before a $\overline{CD}$ lost error is recognized. Otherwise, the negation of $\overline{CD}$ immediately causes the $\overline{CD}$ lost condition to occur.

Note: If the CDS bit in the GSMR_H is set, all $\overline{CD}$ transitions must occur while the receive clock is low.

16.9.10.2 ASYNCHRONOUS PROTOCOLS. In asynchronous protocols, the $\overline{RTS}$ pin is asserted when SCC2 data is loaded into the transmit FIFO and a falling transmit clock occurs. The $\overline{CD}$ and $\overline{CTS}$ pins can be used to control reception and transmission in the same manner as the synchronous protocols. The first bit of data transmission in an asynchronous protocol is the start bit of the first character. In addition, the UART protocol has an option for $\overline{CTS}$ flow control as described in **Section 16.9.15 The SCC2 in UART Mode**.

If $\overline{CTS}$ is already asserted when $\overline{RTS}$ is asserted, transmission begins in two additional bit times. However, if $\overline{CTS}$ is not already asserted when $\overline{RTS}$ is asserted and CTSS = 0, then transmission begins in three additional bit times. If $\overline{CTS}$ is not already asserted when $\overline{RTS}$ is asserted and CTSS = 1 in the GSMR_H, then transmission begins in two additional bit times.

16.9.11 Digital Phase-Locked Loop Operation

Each SCC2 channel includes a digital phase-locked loop (DPLL) that is used to recover clock information from a received datastream. For applications that provide a direct clock source to the serial communication controller, the DPLL can be bypassed if it is programmed to do so in the GSMR_L. The DPLL must not be used when the serial communication controller is programmed to Ethernet and it is optional for other protocols. The DPLL receiver block diagram is illustrated in Figure 16-68 and the transmitter block diagram is in Figure 16-69.

The DPLL can be driven by an external clock or one of the baud rate generator outputs and they should be approximately 8×, 16×, or 32× the data rate, depending on the encoding or decoding preferred. The DPLL uses this clock, along with the datastream, to construct a data clock that can be used as the SCC2 receive and/or transmit clock. In all modes, the DPLL uses the input clock to determine the nominal bit time.

At the beginning of operation, the DPLL is in search mode, whereas the first transition resets the internal DPLL counter and begins DPLL operation. While the counter is counting, the DPLL watches the incoming datastream for transitions and when a transition is detected, the DPLL makes a count adjustment to produce an output clock that tracks the incoming bits.

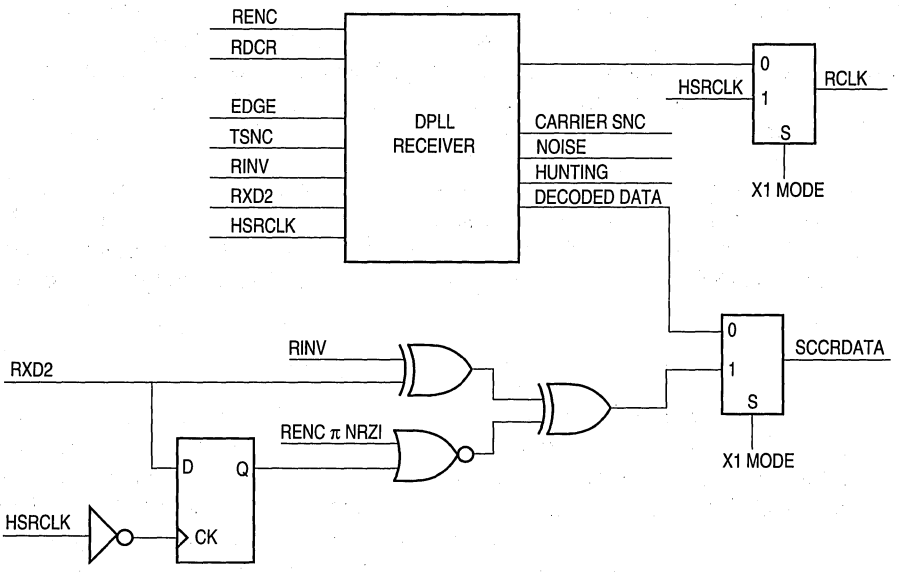


Figure 16-68. DPLL Receiver Block Diagram

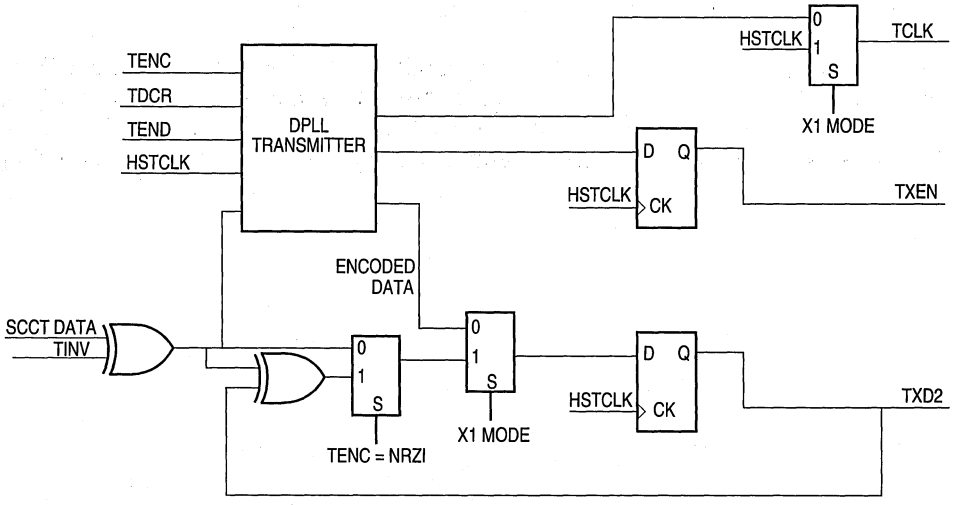


Figure 16-69. DPLL Transmitter Block Diagram

The DPLL has a carrier-sense signal that indicates when there are data transfers on the RXD2 signal. Using the TSNC field of the GSMR_L, this signal is asserted as soon as a transition is detected on RXD2 and it is negated after a programmable number of clocks have been detected with no transitions.

To prevent itself from locking on the wrong edges and to provide fast synchronization, the DPLL should receive a preamble pattern before it receives the data. In some protocols, the preceding flags or syncs are used. However, some protocols require a special pattern, such as alternating ones and zeros. When a transmission occurs, the serial communication controller can generate preamble patterns as programmed in the TPP and TPL bits of the GSMR_L.

Table 16-25. Preamble Patterns for Decoding Methods

DECODING METHOD	PREAMBLE PATTERN	MAXIMUM PREAMBLE LENGTH REQUIRED
NRZI Mark	All zeros	8-bit
NRZI Space	All ones	8-bit
FM0	All ones	8-bit
FM1	All zeros	8-bit
Manchester	Repeating 10's	8-bit
Differential Manchester	All ones	8-bit

In addition, the DPLL can be used to invert the datastream of a reception or transmission. This feature is available in all encodings, including the standard NRZ data format. Also, when the transmitter is idle, the DPLL can either force the TXD2 signal to a high voltage or continue encoding the data supplied to it. The DPLL is used for UART encoding/decoding, which gives you the option of selecting the divide ratio in the UART decoding process (8×, 16×, or 32×). Typically, 16× option is used.

The maximum data rate that can be supported with the DPLL is 3.125MHz when operating with a 25MHz system clock, assuming that the 8× option is chosen (25MHz ÷ 8 = 3.125MHz). Thus, the frequency applied to the CLKx pin or generated by an internal baud rate generator may be up to 25MHz on a 25MHz MPC823, if the DPLL 8×, 16×, or 32× options are used.

Note: The 1:2 ratio of GCLK1 to the serial clock does not apply when the DPLL is used to recover the clock in the 8×, 16×, or 32× modes. Synchronization occurs internally after the receive clock is generated by the DPLL. Therefore, even the fastest DPLL clock generation (the 8× option) easily meets the required 1:2 ratio clocking limit.

16.9.11.1 ENCODING AND DECODING DATA WITH A DPLL. The serial communication controller contains a digital phase-locked loop unit that can be programmed to encode and decode SCC2 data as NRZ, NRZI Mark, NRZI Space, FM0, FM1, Manchester, and Differential Manchester. Examples of the different encoding methods are illustrated in Figure 16-70.

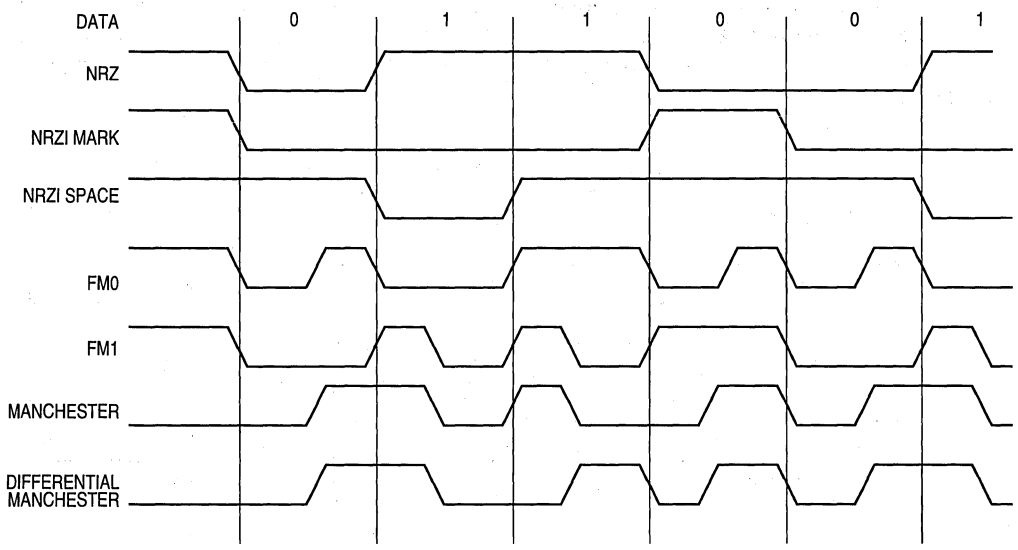


Figure 16-70. DPLL Encoding Examples

If you do not want to use the DPLL, you can choose NRZ coding in the RENC and TENC bits in the GSMR_L. The coding is defined as follows:

- NRZ—A one is represented by a high level for the duration of the bit and a zero is represented by a low level.
- NRZI Mark—A one is represented by no transition at all. A zero is represented by a transition at the beginning of the bit (the level present in the preceding bit is reversed).
- NRZI Space—A one is represented by a transition at the beginning of the bit (the level present in the preceding bit is reversed). A zero is represented by no transition at all.
- FM0—A one is represented by a transition only at the beginning of the bit. A zero is represented by a transition at the beginning of the bit and another transition at the center of the bit.
- FM1—A one is represented by a transition at the beginning of the bit and another transition at the center of the bit. A zero is represented by a transition only at the beginning of the bit.

- Manchester—A one is represented by a high to low transition at the center of the bit. A zero is represented by a low to high transition at the center of the bit. In both cases there may be a transition at the beginning of the bit to set up the level required to make the correct center transition.
- Differential Manchester—A one is represented by a transition at the center of the bit with the opposite direction from the transition at the center of the preceding bit. A zero is represented by a transition at the center of the bit with the same polarity from the transition at the center of the preceding bit.

16.9.12 Clock Glitches

A clock glitch occurs when an input clock signal transitions between a one and zero state two times, in a time period small enough to violate the minimum high or low time specification of the input clock. They also occur when excessive noise is present on a slowly rising or falling signal.

This can be a potential problem for many communication systems. Not only can glitched clocks cause systems to experience errors, but they can also cause undetected the errors. Systems that supply an external clock to a serial channel are often susceptible to glitches from noise, connecting or disconnecting the physical cable from the application board, or excessive ringing on a clock line.

The serial communication controller has a special circuit designed to detect glitches that occur within the system. The glitch circuit is designed to detect glitches that could cause the serial communication controller to transition to the wrong state. This status information can be used to alert the system of a problem at the physical layer. The glitch detect circuit is not a specification test, so if you develop a circuit that does not meet the input clocking specifications for the serial communication controller, erroneous data can be received or transmitted that is not indicated by the glitch detection logic. Conversely, if a glitch indication is signaled, it does not guarantee that erroneous data was received or transmitted. Regardless of whether the DPLL is used, the received clock is passed through a noise filter that eliminates any noise spikes that affect a single sample. This sampling is enabled using the GDE bit of the GSMR_H.

If a spike is detected, a maskable receive or transmit glitched clock interrupt is generated in the event register of the SCC2 channel. Although you can either reset the SCC2 receiver or transmitter or continue operation, the statistics on clock glitches should be kept for later evaluation. The glitch status indication can also be used as a debugging aid during the early phases of prototype testing.

16.9.13 DPLL and Serial Infra-Red Encoder/Decoder

IrDA is a family of specifications intended to facilitate the interconnection of computers and peripherals using a directed half duplex serial infrared physical communications medium. The infra-red data association (IrDA) physical layer standard version 1.1 specifies three modes of operation, each one with a distinct modulation scheme and signaling rate.

- Low speed mode—2.4Kb/s to 115.2Kb/s
- Middle speed mode—0.576 Mb/s or 1.152 Mb/s
- High speed mode—4 Mb/s

Figure 16-71 illustrates how to implement a general infra-red link with the MPC823 using the serial communication controller, IrDA encoder/decoder module, and external IrDA transducer module.

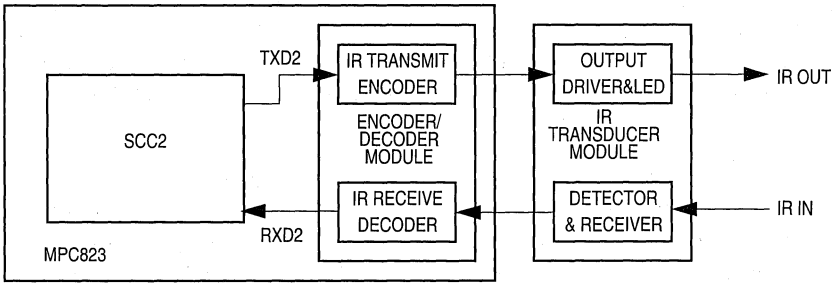


Figure 16-71. Serial IrDA Link

The IrDA DPLL is driven by one of the baud rate generator outputs or by an external clock called a high-speed receive/transmit clock (HSRCLK/HSTCLK) that is used as a reference clock for the IrDA DPLL. In low- and middle-speed modes, the HSRCLK and HSTCLK frequency should be 16x the serial frequency. In high-speed mode, the ratio between the HSRCLK/HSTCLK frequency and the serial frequency depends on the IrDA DPLL mode, which is currently 12x the serial frequency. Therefore, for high-speed mode IrDA operation from an external clock , a 48MHz frequency must be provided. It you are using a source for HSRCLK and HSTCLK, the minimum system frequency must be 48MHz. See **Section 16.9.20.5 Programming Model** for more information.

16.9.14 Disabling the SCC2 On-the-Fly

If you do not need the serial communication controller for a while, you can disable it and reenable it later by following a sequence of steps, which ensures that any buffers currently in use are properly closed and that new data is transferred to or from a new buffer. This sequence must be followed if you are changing a parameter that cannot be dynamically changed. Parameters that are dynamic can be changed immediately. For instance, the internal baud rate generators allow on-the-fly changes, but the DPLL-related bits in the GSMR_L do not.

Note: Modifying parameter RAM does not require that you fully disable the serial communication controller. Refer to the parameter RAM description for details on when parameter RAM values can be modified. If you prefer to disable the USB, SCC2, SMCs, SPI, and the I^2C, use the CPCR to reset the entire communication processor module with a single command.

16.9.14.1 DISABLING THE ENTIRE SCC2 TRANSMITTER. The SCC2 transmitter can be fully disabled or enabled by following these steps:

1. Issue the **STOP TRANSMIT** command to the CPCR if the serial communication controller is currently transmitting data. It should stop smoothly. If the serial communication controller is not transmitting, then you do not need the **STOP TRANSMIT** command. Furthermore, if you overwrite the TBPTR or execute the **INIT TX PARAMETERS** command, the **STOP TRANSMIT** command is not required.
2. Clear the ENT bit in the GSMR_L. The SCC2 transmitter is now disabled and put in a reset state.
3. Make modifications to the SCC2 transmit parameters or to the parameter RAM. If you want to switch protocols or restore the SCC2 transmit parameters to their initial state, issue the **INIT TX PARAMETERS** command to the CPCR.
4. Issue the **RESTART TRANSMIT** command to the CPCR. This command is required if the **INIT TX PARAMETERS** command was not issued in step 3.
5. Set the ENT bit in the GSMR_L. Transmission begins using the TX buffer descriptor that the TBPTR points to as soon as the R bit is set in the TX buffer descriptor.

16.9.14.2 DISABLING PART OF THE SCC2 TRANSMITTER. To reinitialize the SCC2 transmitter to the state it was in after reset, follow this short sequence of steps:

1. Clear the ENT bit in the GSMR_L.
2. Issue the **INIT TX PARAMETERS** command and make any additional modifications.
3. Set the ENT bit in the GSMR_L.

16.9.14.3 DISABLING THE ENTIRE SCC2 RECEIVER. The SCC2 receiver can be fully disabled or enabled by following these steps:

1. Clear the ENR bit in the GSMR_L. The SCC2 receiver is now disabled and put in a reset state.
2. Make modifications to the SCC2 receive parameters or to the parameter RAM. If you want to switch protocols or restore the SCC2 receive parameters to their initial state, issue the **INIT RX PARAMETERS** command.
3. Issue the **ENTER HUNT MODE** command in the CPCR. This command is required if the **INIT RX PARAMETERS** command was not issued in step 2.
4. Set the ENR bit in the GSMR_L. Reception begins using the RX buffer descriptor that the RBPTR points to if the E bit is set in the TX buffer descriptor.

16.9.14.4 DISABLING PART OF THE SCC2 RECEIVER. To reinitialize the SCC2 receiver to the state it was in after reset, follow this short sequence of steps:

1. Clear the ENR bit in the GSMR_L.
2. Issue the **INIT RX PARAMETERS** command in the CPCR and make any additional modifications.
3. Set the ENR bit in the GSMR_L.

16.9.14.5 SWITCHING PROTOCOLS. Sometimes you may want to switch the protocol that the serial communication controller is executing without resetting the board. You can do this by using one command and the following sequence of steps:

1. Clear the ENT and ENR bits in the GSMR_L.
2. Issue the **INIT TX AND RX PARAMS** command in the CPCR. This command initializes both the transmit and receive parameters. Any additional modifications can be made in the GSMR_L.
3. Set the ENT and ENR bits in the GSMR_L. The serial communication controller is enabled with the new protocol.

Tip: You can save power by clearing the ENT and ENR bits of the serial communication controller.

16.9.15 The SCC2 in UART Mode

Many applications need a simple method of sending low-speed data between pieces of equipment. The universal asynchronous receiver transmitter (UART) protocol is the de-facto standard for such communication. The term asynchronous is used because it is not necessary to send clocking information along with the data being sent. UART links are character-oriented. Asynchronous links are used to connect terminals and other computer equipment together. Even in applications where synchronous communication is required, the UART is often used for a local debugging port to run board debugger software. The character format of the UART protocol is illustrated in Figure 16-72.

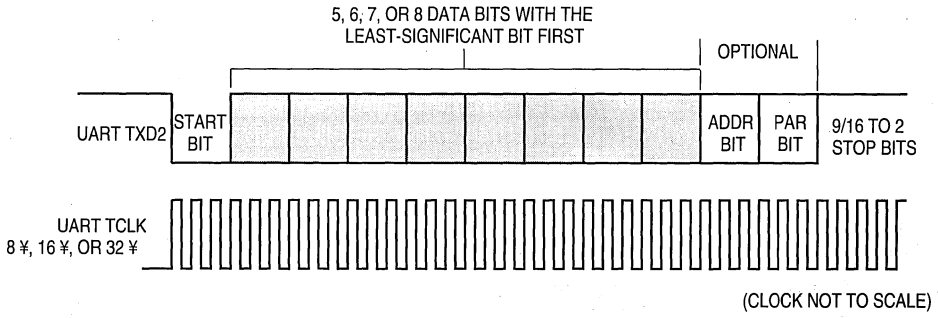


Figure 16-72. UART Character Format

Since the transmitter and receiver operate asynchronously, there is no need to connect the transmit and receive clocks. Instead, the receiver oversamples the incoming datastream (usually by a factor of 16) and uses some of these samples to determine the bit value. Traditionally, the middle three samples of the 16 samples are used. Two UARTs can communicate using a system like this if parameters, such as the parity scheme and character length, are the same for both the transmitter and receiver.

When data is not transmitted in the UART protocol, a continuous stream of ones is transmitted called the idle condition. Since the start bit is always a zero, the receiver can detect when real data is once again present on the line. UART specifies an all-zeros character called a break, which is used to abort a character transfer sequence.

Many different protocols have been defined using asynchronous characters, but the most popular of these is the RS-232 standard, which specifies baud rates, handshaking protocols, and mechanical/electrical details. Another popular standard using the same character format is RS-485, which defines a balanced line system allowing longer cables than RS-232 links. Synchronous protocols are sometimes defined to run over asynchronous links. Other protocols like Profibus extend the UART protocol to include LAN-oriented features such as token passing.

All the standards provide handshaking signals, but some systems require just three physical lines—transmit data, receive data, and ground. Many proprietary standards have been built around the asynchronous character frame and some even implement a multidrop configuration. In multidrop systems, more than two stations can be present on a network, each one with a specific address. Frames made up of many characters can be broadcast with the first character acting as a destination address. To accommodate this, the UART frame is extended one bit to distinguish between an address character and the normal data characters. Additionally, a synchronous form of the UART protocol exists where start and stop bits are still present, but because a clock is provided with each bit the oversampling technique is not required. This mode is called isochronous operation or synchronous UART.

By appropriately setting the GSMR_L, the SCC2 channel can be configured to function in UART mode, which provides standard serial I/O using asynchronous character-oriented (start-stop) protocols with RS-232 C-type lines. The SCC2 in UART mode, also called the SCC2 UART controller, can be used to communicate with any existing RS-232 type of device and to provide a port for serial communication to other microprocessors and terminals (either locally or via modems). It includes facilities for communication using standard asynchronous bit rates and protocols. The SCC2 in UART mode supports a multidrop mode for master/slave operation with wake-up capability on both the idle signal and address bit. This mode also supports a synchronous mode of operation where a clock must be provided with each bit received. It transmits data from memory (either internal or external) to the TXD2 signal and receives data from the RXD2 signal into memory. In synchronous UART mode, the clock must also be supplied and it can be generated internally or externally. Modem lines are supported via the port B and C pins. The SCC2 in UART mode consists of separate transmit and receive sections whose operations are asynchronous with the core.

16.9.15.1 FEATURES

The following list summarizes the main features of the SCC2 in UART mode:

- Flexible message-oriented data structure
- Implements synchronous and asynchronous UART
- Multidrop operation
- Receiver wake-up on idle line or address mode
- Eight control character comparisons
- Two address comparisons
- Maintenance of four 16-bit error counters
- Received break character length indication
- Programmable data length (5–8 bits)
- Programmable 1 to 2 stop bits in transmission
- Capable of reception without a stop bit
- Programmable fractional stop bit length
- Even, odd, force, or no parity generation

- Even, odd, force, or no parity check
- Frame error, noise error, break, and idle detection
- Transmit preamble and break sequences
- Freeze transmission option with low-latency stop

16.9.15.2 NORMAL ASYNCHRONOUS MODE. In normal asynchronous mode, the receive shift register receives the incoming data on the RXD2 pin. The control bits in the PSMR–SCC2 UART register define the length and format of the UART character and each bit is received in the following order:

1. Start bit
2. 5–8 data bits (LSB first)
3. Address/data bit (optional)
4. Parity bit (optional)
5. Stop bits

The receiver uses a clock 8×, 16×, or 32× faster than the baud rate and samples each bit of the incoming data three times around its center. The value of the bit is determined by the majority of those samples and if they do not all agree, a noise indication counter is incremented. When a complete byte has been clocked in, the contents of the shift register are transferred to a UART receive data buffer. If there is an error in this character, the appropriate error bits are set by the communication processor module.

The SCC2 UART controller can receive fractional stop bits. The next character's start bit can begin any time after the three middle samples are taken. The UART transmit shift register transmits the outgoing data on the TXD2 pin. Data is then clocked synchronously with the transmit clock, which may have either an internal or external source. The bit transmission order is LSB first, but only the data portion of the UART frame is actually stored in the data buffers. The start and stop bits are always generated and stripped by the SCC2 UART controller. The parity bit can also be generated in transmission and checked during reception and although it is not stored in the data buffer, its value can be inferred from the reporting mechanism of the data buffer. Similarly, the optional address bit is not stored in the transmit or receive data buffer, but is implied from the buffer descriptor itself. Parity is generated and checked for the address bit. The RFW bit of the GSMR_H must be set for an 8-bit receive FIFO.

16.9.15.3 SYNCHRONOUS MODE. In synchronous mode, the SCC2 UART controller uses a 1× data clock for timing. The receive shift register receives the incoming data on the RXD2 pin synchronously to the clock. The length and format of the serial word in bits are defined by the control bits in the PSMR–SCC2 UART register in the same way they were in asynchronous mode. When a complete byte has been clocked in, the contents of the shift register are transferred to a UART receive data buffer. If there is an error in this character, then the appropriate error bits are set by the communication processor module.

The UART transmit shift register transmits the outgoing data on the TXD2 pin. Data is then clocked synchronously with the transmit clock, which can have an internal or external source. The RFW bit in the GSMR_H must be set for an 8-bit receive FIFO.

16.9.15.4 SCC2 UART PARAMETER RAM MEMORY MAP. When configured to operate in UART mode, the serial communication controller overlays the structure used in Table 16-24 with the UART parameters described in Table 16-26.

Table 16-26. SCC2 UART Parameter RAM Memory Map

ADDRESS	NAME	WIDTH	DESCRIPTION
SCC2 Base + 30	RES	Word	Reserved
SCC2 Base + 34	RES	Word	Reserved
SCC2 Base + 38	**MAX_IDL**	Half-word	MaximumIdle Characters
SCC2 Base + 3A	IDLC	Half-word	Temporary idle Counter
SCC2 Base + 3C	**BRKCR**	Half-word	Break Count Register (Transmit)
SCC2 Base + 3E	**PAREC**	Half-word	Receive Parity Error Counter
SCC2 Base + 40	**FRMEC**	Half-word	Receive Framing Error Counter
SCC2 Base + 42	**NOSEC**	Half-word	Receive Noise Counter
SCC2 Base + 44	**BRKEC**	Half-word	Receive Break Condition Counter
SCC2 Base + 46	BRKLN	Half-word	Last Received Break Length
SCC2 Base + 48	**UADDR1**	Half-word	UART Address Character 1
SCC2 Base + 4A	**UADDR2**	Half-word	UART Address Character 2
SCC2 Base + 4C	RTEMP	Half-word	Temp Storage
SCC2 Base + 4E	**TOSEQ**	Half-word	Transmit Out-of-Sequence Character
SCC2 Base + 50	**CHARACTER1**	Half-word	Control Character 1
SCC2 Base + 52	**CHARACTER2**	Half-word	Control Character 2
SCC2 Base + 54	**CHARACTER3**	Half-word	Control Character 3
SCC2 Base + 56	**CHARACTER4**	Half-word	Control Character 4
SCC2 Base + 58	**CHARACTER5**	Half-word	Control Character 5
SCC2 Base + 5A	**CHARACTER6**	Half-word	Control Character 6
SCC2 Base + 5C	**CHARACTER7**	Half-word	Control Character 7
SCC2 Base + 5E	**CHARACTER8**	Half-word	Control Character 8
SCC2 Base + 60	**RCCM**	Half-word	Receive Control Character Mask
SCC2 Base + 62	RCCRP	Half-word	Receive Control Character Register
SCC2 Base + 64	RLBC	Half-word	Receive Last Break Character

NOTE: You are only responsible for initializing the items in bold.
SCC2 base = (IMMR & 0xFFFF0000) + 0x3D00.
All references to registers in the parameter RAM table are actually implemented in the dual-port RAM area as a memory-based register.

- MAX_IDL—Once a character is received, the SCC2 UART controller begins counting any idle characters that are received. If a MAX_IDL number of idle characters is received before the next data character, an idle timeout occurs and the buffer is closed. This, in turn, can produce an interrupt request to the core to receive the data from the buffer. Thus, MAX_IDL provides a convenient way to demarcate frames in UART mode. To disable the MAX_IDL feature, simply program it to 0x0000. An idle character is calculated the following number of bit times: 1 + data length (5, 6, 7, or 8) + 1 (if parity bit is used) + number of stop bits (1 or 2). For example, for 8 data bits, no parity, and 1 stop bit, the character length is 10 bits.
- IDLC—The RISC microcontroller uses this value to store the current idle counter value in the MAX_IDL timeout process. IDLC is a down-counter and you do not need to initialize or access it.
- BRKCR—The SCC2 UART controller sends a break character sequence whenever a **STOP TRANSMIT** command is issued to the CPCR. The number of break characters sent by the SCC2 UART controller is determined by the value in BRKCR. For 8 data bits, no parity, 1 stop bit, and 1 start bit, each break character is 10 bits long and consists of all zeros.
- PAREC, FRMEC, NOSEC, and BRKEC—You must initialize these 16-bit (modulo–2^{16}) counters. When the following conditions occur, they are incremented by the RISC microcontroller.
  - ❑ PAREC counts received parity errors.
  - ❑ FRMEC counts received characters with framing errors.
  - ❑ NOSEC counts received characters with noise errors.
  - ❑ BRKEC counts the number of break conditions that occur on the RX pin. Notice that one break condition can last for hundreds of bit times, yet this counter is incremented only once during that period.
- BRKLN—This value is used to store the length of the last break character that is received and is as long as the break. For example, if the receive pin is low for 20 bit times, BRKLN shows the value 0x0010. BRKLN is accurate to within one character unit of bits. For 8 data bits, no parity, 1 stop bit, and 1 start bit, BRKLN is accurate to within 10 bits.
- UADDR1, UADDR2—In multidrop mode, the SCC2 UART controller provides automatic address recognition for two addresses. In this case, you program the lower order bytes of UADDR1 and UADDR2 with the two preferred addresses.
- TOSEQ—This value is used to transmit out-of-sequence characters in the transmit stream. Using this field, the preferred characters can be inserted into the transmit FIFO without affecting any transmit buffer that might currently be in progress.
- CHARACTER1 to CHARACTER 8—These characters define the receive control characters on which interrupts can be generated.
- RCCM—This value is used to mask the comparison of the CHARACTER1 to CHARACTER 8 parameters so that classes of control characters can be defined. A one enables the bit comparison and a zero masks it.

- RCCRP—This value is used to hold the value of any control character that is not written to the data buffer.
- RLBC—This entry is used in synchronous UART when the RZS bit is set in the PSMR–UART and contains the actual pattern of the last break character. By counting the zeros in this entry, the core can measure the break length to a bit resolution. You read RLBC by counting the number of zeros written, starting at bit 15 continuing to the point where the first one was written. Therefore, RLBC = 001xxxxxxxxxxxx (binary) indicates two zeros and RLBC = 1xxxxxxxxxxxxxx (binary) indicates no zeros.

16.9.15.5 PROGRAMMING THE SCC2 IN UART MODE. The SCC2 UART controller uses the same data structure as the other modes and supports multibuffer or multidrop operation. You can program the SCC2 UART controller to reject messages that are not destined for a programmable address (multidrop mode). You can also program the SCC2 UART controller to accept or reject control characters. If a control character is rejected, an interrupt can be generated. The receive character can be accepted using a receive character mask value.

The SCC2 UART controller enables you to transmit break and preamble sequences. Overrun, parity, noise, and framing errors are reported via the buffer descriptor table and/or error counters. An indication of the signal line status is reported in the status register and a maskable interrupt is generated when the status changes. In its simplest form, the SCC2 UART controller functions in a character-oriented environment, in which each character is transmitted with the stop bits and parity and received into separate 1-byte buffers. A maskable interrupt is generated when a buffer is received.

Using linked buffers, many applications try to take advantage of the message-oriented capabilities that the serial communication controller supports in UART mode. Data is handled in a message-oriented environment which means that you can work on entire messages rather than operating on a character-by-character basis. Also, a message can span several linked buffers. For example, before handling the input data, a terminal driver may want to wait until you type an end-of-line character rather than be interrupted when a character is received.

As another example, when transmitting ASCII files, the data can be transferred as messages ending on the end-of-line character. Each message could be both transmitted and received as a linked list of buffers without any intervention from the core, which makes it easy to program and saves processor overhead. Before reception you can define up to eight control characters and each control character can be configured to designate the end of a message or generate a maskable interrupt without being stored in the data buffer. The latter option is useful when flow control characters such as XON or XOFF need to alert the core but do not belong to the received message.

16.9.15.6 SCC2 UART COMMANDS. You can program the CPM command register (CPCR) with the following commands to transmit data.

- **STOP TRANSMIT**—After a hardware or software reset and a channel is enabled in the PSMR–SCC2 UART register, the channel is in transmit enable mode and starts polling the first buffer descriptor in the table every eight transmit clocks. This command disables the transmission of characters on the transmit channel and if it is received by the SCC2 UART controller while a message is transmitting, the message is aborted. The SCC2 UART controller finishes transmitting data that is already transferred to its FIFO and then stops. The TBPTR is not incremented, as shown in **Section 16.9.7 SCC2 Parameter RAM Memory Map**. The UART transmitter transmits a programmable number of break sequences and starts transmitting idles. The number of break sequences (which can be zero) should be written to the BRKCR before this command is given to the SCC2 UART controller.
- **GRACEFUL STOP TRANSMIT**—This command is used to stop transmitting smoothly, rather than abruptly. It is similar to the way the **STOP TRANSMIT** command finishes. It stops after the current buffer has completed transmission or immediately if there is no buffer being transmitted. The GRA bit in the SCCE–UART register is set once this transmission stops. Then the UART transmit parameters, including the buffer descriptors, can be modified. The TBPTR points to the next TX buffer descriptor in the table. Transmission begins once the R bit of the next buffer descriptor is set and the **RESTART TRANSMIT** command is issued.
- **RESTART TRANSMIT**—This command enables characters to be transmitted on the transmit channel. The SCC2 UART controller expects this command after it disables the channel in its PSMR–SCC2 UART register, after a **STOP TRANSMIT** command, after a **GRACEFUL STOP TRANSMIT** command, or after a transmitter error. The SCC2 in UART mode resumes transmission from the current TBPTR in the channel's TX buffer descriptor table.
- **INIT TX PARAMETERS**—This command initializes all transmit parameters in the serial channel's parameter RAM to their reset state and should only be issued when the transmitter is disabled. Notice that the **INIT TX AND RX PARAMS** command can be used to reset both the transmit and receive parameters.

You can program the CPCR with the following commands to receive data.

- **ENTER HUNT MODE**—After the hardware or software is reset and a channel is enabled in the PSMR–SCC2 UART register (described in **Section 16.9.15.15 SCC2 UART Mode Register**), the channel is in receive enable mode and uses the first buffer descriptor in the table. This command forces the SCC2 UART controller to close the current RX buffer descriptor if it is being used and enter hunt mode. The SCC2 UART controller continues receiving the next buffer descriptor if a message is in progress. In the multidrop hunt mode, the SCC2 UART controller continually scans the input datastream for the address character. When it is not in multidrop mode, it waits for the idle sequence (one character of idle) and does not lose any data that was in the receive FIFO when this command was executed.

- **CLOSE RX BD**—This command forces the serial communication controller to close the RX buffer descriptor if it is currently being used and it uses the next buffer descriptor for any subsequently received data. If the serial communication controller is not in the process of receiving data, no action is taken.

Note: The **CLOSE RX BD** command in the SCC2 UART controller functions the same as the **ENTER HUNT MODE** command, except for one difference. **CLOSE RX BD** does not need an idle character to be present on the line before it continues receiving.

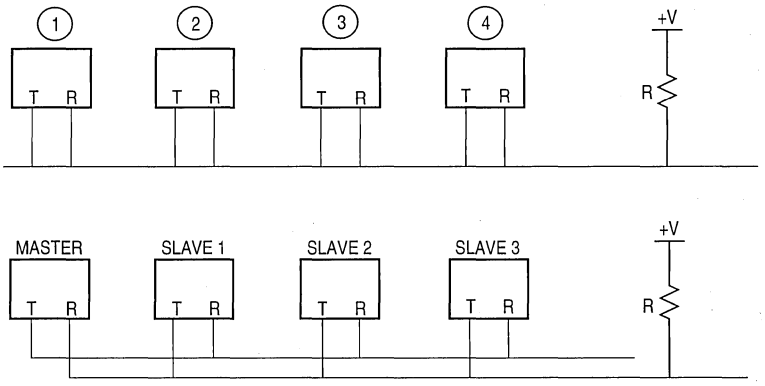
- **INIT RX PARAMETERS** —This command initializes all receive parameters in this serial channel's parameter RAM to their reset state and should only be issued when the receiver is disabled. Notice that the **INIT TX AND RX PARAMS** command can be used to reset the receive and transmit parameters.

16.9.15.7 RECOGNIZING ADDRESSES IN SCC2 UART MODE. In multidrop systems, more than two stations can be present on a network and each one can have a specific address. Figure 16-73 illustrates two examples of this configuration. Frames made up of many characters can be broadcast as long as the first character is the destination address. To achieve this, the UART frame is extended by one bit to distinguish between an address character and the data characters. The SCC2 UART controller can be configured to operate in a multidrop environment that supports the following two modes:

- Automatic Multidrop Mode—The SCC2 UART controller automatically checks the incoming address character and accepts the data following it, but only if the address matches one of two preset values.
- Manual Multidrop Mode—The SCC2 UART controller receives all characters. An address character is always written to a new buffer and it can be followed by data characters.

The SCC2 UART controller has two 16-bit address registers that support address recognition—UADDR1 and UADDR2. The upper 8 bits of these registers should be written with zero because only the lower 8 bits are used. In automatic mode, the incoming address is checked against UADDR1 and UADDR2 and when a match is made, the AM bit in the buffer descriptor is set to indicate the matched address character and the data following it is written to the data buffers.

Note: For characters less than 8 bits, the MSBs should be zero.



UADDR1
UADDR2

TWO 8-BIT ADDRESSES CAN BE AUTOMATICALLY RECOGNIZED IN EITHER CONFIGURATION.

PAODR

CHOOSE WIRED-OR OPERATION IN THE PORT A OPEN-DRAIN REGISTER TO ALLOW MULTIPLE TRANSMIT PINS TO BE DIRECTLY CONNECTED.

Figure 16-73. Two UART Multidrop Mode Configuration Examples

16.9.15.8 SCC2 UART CONTROL CHARACTERS. The SCC2 UART controller can recognize special control characters that can be used in a message-oriented environment. You can define a maximum of eight control characters in the control characters table. Each character can either be written to the receive buffer or rejected. If it is rejected, the character is written to the received control character register in internal RAM and a maskable interrupt is generated. This method is used to let you know that the control characters (XON or XOFF) that are not part of the received messages have arrived. The SCC2 UART controller uses a table of 16-bit entries to support control character recognition and each entry consists of the control character, a valid bit, and a reject character bit.

	0	1	2	3	4	5	6	7	8 9 10 11 12 13 14 15
SCC2 BASE + 50	E	R	—	—	—	—	—	—	CHARACTER1
SCC2 BASE + 52	E	R	—	—	—	—	—	—	CHARACTER2
SCC2 BASE + 54	E	R	—	—	—	—	—	—	CHARACTER3
•									
•									
•									
SCC2 BASE+ 5E	E	R	—	—	—	—	—	—	CHARACTER8
SCC2 BASE+ 60	1	1	—	—	—	—	—	—	RCCM
SCC2 BASE + 62	—	—	—	—	—	—	—	—	RCCRP

E—End of Table

In tables with eight control characters, this bit is always zero.

0 = This entry is valid. The lower 8 bits are checked against the incoming character.
1 = This entry is invalid and must be the last entry in the control characters table.

R—Reject Character

0 = The character is not rejected, but is written into the receive buffer. The buffer is then closed and a new receive buffer is used if there is more data in the message. A maskable interrupt is generated.
1 = If this character is recognized, it is not written to the receive buffer. Instead, it is written to the RCCRP register and a maskable interrupt is generated. The current buffer is not closed when a control character is received with the R bit set.

CHARACTER1–CHARACTER8—Control Character Values 1–8

These fields define control characters that will be compared to the incoming character. For characters smaller than 8 bits, the most-significant bit should be zero.

RCCM—Received Control Character Mask

This value is used to mask the comparison of the CHARACTER1–CHARACTER8 fields. The lower 8 bits of the RCCM correspond to the lower 8 bits of CHARACTER1–CHARACTER8 and are decoded as follows:

0 = Mask this bit when the incoming character and CHARACTER1–CHARACTER8 fields are compared.
1 = The address comparison on this bit proceeds normally and no masking occurs.

Note: Bits 0 and 1 of RCCM must be set or erratic operation will occur during the control character recognition process.

RCCRP—Received Control Character Register

When a control character match is made and the R bit is set, the SCC2 UART controller writes the control character into the RCCRP and generates a maskable interrupt. The core must process the interrupt and read the RCCRP before a second control character arrives. If this does not occur, the SCC2 UART controller overwrites the first control character.

16.9.15.9 WAKE-UP TIMER. By issuing the **ENTER HUNT MODE** command, you can temporarily disable the UART receiver and make it inactive until an idle or address character is recognized, depending on how the UM field is set in the PSMR–SCC2 UART register. See **Section 16.9.15.15 SCC2 UART Mode Register** for more information.

If the SCC2 UART controller is still in the process of receiving a message that you have already decided to discard, you can abort its reception by issuing the **ENTER HUNT MODE** command. When the message is finished, the UART receiver is reenabled by finding the idle line or the address bit of the next message, depending on how the UM field is set. When the receiver is in sleep mode and receives a break sequence, it increments the BRKEC counter and generates an interrupt if the BRKE or BRKS bits are enabled in the SCCM–UART register. Refer **Section 16.9.15.16 SCC2 UART Receive Buffer Descriptors** for more information about the type of receive interrupt that is registered.

16.9.15.10 BREAK SUPPORT. The SCC2 UART controller provides flexible break support to the receiver. Transmitting out-of-sequence characters is also supported by the SCC2 UART controller and is normally used for the transmission of flow control characters like XON or XOFF. This procedure is performed using the TOSEQ entry in the SCC2 UART parameter RAM.

The SCC2 UART controller polls the TOSEQ character whenever the transmitter is enabled for UART operation, including during a UART freeze operation, UART buffer transmission, and when no buffer is ready for transmission. TOSEQ is transmitted at a higher priority than the other characters in the transmit buffer, but does not preempt characters already in the transmit FIFO. This means that the XON or XOFF character may not be transmitted for eight or four character times. To reduce this latency, the TFL bit in the GSMR_H should be set to decrease the FIFO size to one character prior to enabling the SCC2 transmitter.

TOSEQ

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RES		REA	I	CT	RES		A	CHARSEND							
RESET	0		0	0	0	0		0	0							
R/W	R/W		R/W	R/W	R/W	R/W		R/W	R/W							
ADDR	SCC2 BASE + 0x4E															

Bits 0–1 and 5–6—Reserved

These bits are reserved and should be set to 0.

REA—Ready

This bit is set by the core when the character is ready for transmission and remains 1 while the character is being transmitted. The communication processor module clears this bit after transmission.

I—Interrupt

If this bit is set, the character in the CHARSEND field is transmitted. The TX bit is then set in the SCCE–UART register and the core may be interrupted.

CT—Clear-to-Send Lost

This status bit indicates that the $\overline{CTS}$ signal was negated when this character was transmitted. If negation occurs, the CT bit in the UART transmit buffer descriptor is also set. The DIAG field in the GSMR_L controls whether the $\overline{CTS}$ signal is monitored by the serial communication controller.

Note: If the $\overline{CTS}$ signal is negated during transmission and the communication processor module transmits this character in the middle of a buffer transmission, the $\overline{CTS}$ signal could actually have been negated either during this character's transmission or during a buffer character's transmission. In either case, the communication processor module sets the CT bit both here and in the TX buffer descriptor status word.

A—Address

0 = In multidrop mode, the character being sent is a data character.
1 = In multidrop mode, the character being sent is address character.

CHARSEND—Character Send

This field contains the character to be transmitted. Any 5-, 6-, 7-, or 8-bit character value can be transmitted in accordance with the SCC2 UART configuration. The character is in the least-significant bits of CHARSEND. This value can be modified only while the REA bit is cleared.

16.9.15.11 SENDING A BREAK. A break is an all-zeros character without a stop bit and you can send it by issuing the **STOP TRANSMIT** command. The SCC2 UART controller finishes transmitting any outstanding data, sends a programmable number of break characters according to the BRKCR, and then reverts to idle or sends data if the **RESTART TRANSMIT** command was given before completion. When the break code is complete, the transmitter sends at least one high bit before transmitting anymore data to guarantee a valid start bit will be recognized. The break characters do not preempt characters already in the transmit FIFO, which means that the break character may not be transmitted for eight or four character times. To reduce this latency, set the TFL bit in the GSMR_H so that the FIFO size will be reduced to one character before the SCC2 transmitter is enabled.

16.9.15.12 SENDING A PREAMBLE. A preamble sequence is a convenient way for you to ensure that a line is idle before you start a new message. The preamble sequence length is constructed of consecutive ones of one character length. If the P bit in the UART transmit buffer descriptor is set, the serial communication controller sends a preamble sequence before transmitting that data buffer. For example, for 8 data bits, no parity, 1 stop bit, and 1 start bit, a preamble of 10 ones is sent before the first character in the buffer.

16.9.15.13 FRACTIONAL STOP BITS. You can program the asynchronous UART transmitter to transmit fractional stop bits. Using four bits in the SCC2 UART data synchronization register (DSR–SCC2 UART), you can program the length of the last stop bit to be transmitted. These bits can be modified at any time and if two stop bits are transmitted, only the second one is affected. Idle characters are always transmitted as full-length characters.

DSR–SCC2 UART

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	X	FSB				X	X	X	X	X	X	X	X	X	X	X
RESET	0	1				1	1	0	0	1	1	1	1	1	1	0
R/W	R/W	R/W				R										
ADDR	(IMMR & 0xFFF0000) + 0xA2E															

NOTE: X = "Don't Care".

FSB—Fractional Stop Bit

The value you programmed into the TDCR determines which set of values to use here. When the serial communication controller is in UART mode with 32× oversampling, the FSB field is decoded as follows in the DSR–SCC2 UART:

1111 = Last transmitted stop bit 32/32. Default value after reset.
1110 = Last transmitted stop bit 31/32.
1101 = Last transmitted stop bit 30/32.
1100 = Last transmitted stop bit 29/32.
1011 = Last transmitted stop bit 28/32.
1010 = Last transmitted stop bit 27/32.
1001 = Last transmitted stop bit 26/32.
1000 = Last transmitted stop bit 25/32.
0111 = Last transmitted stop bit 24/32.
0110 = Last transmitted stop bit 23/32.
0101 = Last transmitted stop bit 22/32.
0100 = Last transmitted stop bit 21/32.
0011 = Last transmitted stop bit 20/32.
0010 = Last transmitted stop bit 19/32.
0001 = Last transmitted stop bit 18/32.
0000 = Last transmitted stop bit 17/32.

For 16× oversampling, the FSB field is decoded as follows:

1111 = Last transmitted stop bit 16/16. Default value after reset.
1110 = Last transmitted stop bit 15/16.
1101 = Last transmitted stop bit 14/16.
1100 = Last transmitted stop bit 13/16.
1011 = Last transmitted stop bit 12/16.
1010 = Last transmitted stop bit 11/16.
1001 = Last transmitted stop bit 10/16.
1000 = Last transmitted stop bit 9/16.
0xxx = Invalid. Do not use.

For 8× oversampling, the FSB field is decoded as follows:

1111 = Last transmitted stop bit 8/8. Default value after reset.
1110 = Last transmitted stop bit 7/8.
1101 = Last transmitted stop bit 6/8.
1100 = Last transmitted stop bit 5/8.
10xx = Invalid. Do not use.
01xx = Invalid. Do not use.
00xx = Invalid. Do not use.

The SCC2 UART receiver can always receive fractional stop bits. The next character's start bit can begin at any time after the three middle samples of the stop bit have been taken.

16.9.15.14 SCC2 UART CONTROLLER ERRORS. The SCC2 UART controller reports character reception and transmission error conditions via the channel buffer descriptors, the error counters, and the SCCE–UART register. The modem interface lines can be monitored by the port B and C pins. The following transmission error can be detected by the SCC2 UART controller.

- $\overline{\text{CTS}}$ Lost During Character Transmission —When this error occurs, the channel stops transmission after finishing transmission of the current character from the buffer. The channel then sets the CT bit in the TX buffer descriptor and generates the TX interrupt if it is not masked. The channel resumes transmission after the **RESTART TRANSMIT** command is issued and the $\overline{\text{CTS}}$ pin is asserted.

Note: Using $\overline{\text{CTS}}$ signal, the SCC2 UART controller also offers an asynchronous flow control option that does not generate an error. Refer to the FLC bit of the PSMR–SCC2 UART register description in **Section 16.9.15.15 SCC2 UART Mode Register** for information about flow control.

The following reception errors can be detected by the SCC2 UART controller:

- Overrun Error—This error occurs when data is moved from the receiver FIFO to the data buffer after the first byte is received. If a receiver FIFO overrun occurs, the channel writes the received character into the internal FIFO and over the previously received character. The channel then writes the received character to the buffer, closes it, sets the OV bit in the RX buffer descriptor, and generates the RX interrupt if it is enabled. In automatic multidrop mode, the receiver enters hunt mode immediately.
- $\overline{CD}$ Lost During Character Reception Error—If this error occurs and the channel is using this pin to automatically control reception, the channel terminates character reception, closes the buffer, sets the CD bit in the RX buffer descriptor, and generates the RX interrupt if it is enabled. This error has the highest priority. The last character in the buffer is lost and other errors are not checked. In automatic multidrop mode, the receiver enters the hunt mode immediately.
- Parity Error—When a parity error occurs, the channel writes the received character to the buffer, closes the buffer, sets the PR bit in the RX buffer descriptor, and generates the RX interrupt if it is enabled. The channel also increments the PAREC counter. In automatic multidrop mode, the receiver enters hunt mode immediately.
- Noise Error—The SCC2 UART controller detects a noise error when three different samples are taken on every bit. When this error occurs, the channel writes the received character to the buffer, proceeds normally, but increments the noise error.

Note: A noise error will not occur when the SCC2 UART controller is in synchronous mode.

- Idle Sequence Receive Error—When the SCC2 UART controller receiver receives all ones in the receive buffer (idle sequence), the channel counts the number of consecutive idle characters that were received. If the count reaches the value programmed into MAX_IDL, the buffer is closed and an RX interrupt is generated. If no receive buffer is open, this event does not generate an interrupt or any status information. The internal idle counter (IDLC) is reset every time a character is received.

Note: To disable the idle sequence function, set the MAX_IDL value to zero.

- Framing Error—The SCC2 UART controller gets this error when it receives a character with no stop bit. All framing errors are reported by the SCC2 UART controller, regardless of its mode. When this error occurs, the channel writes the received character to the buffer, closes it, sets the FR bit in the RX buffer descriptor, and generates the RX interrupt if it is enabled. The channel also increments FRMEC. When this error occurs, parity is not checked for this character. In automatic multidrop mode, the receiver immediately enters hunt mode. If the RZS and SYN bits are set in the PSMR–SCC2 UART register when the SCC2 UART controller is in synchronous mode, the receiver reports all framing errors, but continues reception if the unexpected zero is really the start bit of the next character. If RZS is set, your software may not consider a reported SCC2 UART framing error as a true framing error, unless two or more framing errors occur within a short period of time.
- Break Sequence Error—The SCC2 UART controller provides flexible break support to the receiver. When the first break sequence is received, the SCC2 UART controller increments the BRKEC and issues the break start event in the SCCE–UART register, which can generate an interrupt if it is enabled. The SCC2 UART controller then measures the break length and, when the break sequence is complete, writes the length to the BRKLN register. After the first one is received, the SCC2 UART controller also issues the break end event in the SCCE–UART register, which can generate an interrupt if it is enabled. If the SCC2 UART controller was in the process of receiving characters when the break was received, it also closes the receive buffer, sets the BR bit in the RX buffer descriptor, and writes the RX bit in the SCCE–UART register, which can generate an interrupt if it is enabled. If the RZS bit is set in the PSMR–SCC2 UART register when the SCC2 UART controller is in synchronous mode, then a break sequence is detected after only two successive break characters are received.

16.9.15.15 SCC2 UART MODE REGISTER. When the SCC2 is in UART mode, the 16-bit, memory-mapped, read/write protocol-specific mode register is referred to as the SCC2 UART mode register (PSMR–SCC2 UART). Since each protocol has specific requirements, the PSMR bits are different for each implementation. Many of the bits can be modified on-the-fly while the receiver and transmitter are enabled.

PSMR–SCC2 UART

BIT	0	1	2 3	4 5	6	7	8	9	10	11	12 13	14 15
FIELD	FLC	SL	CL	UM	FRZ	RZS	SYN	DRT	RES	PEN	RPM	TPM
RESET	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA28											

FLC—Flow Control

0 = Normal operation. The GSMR_x and port C registers determine the mode of the $\overline{\text{CTS}}$ pin.
1 = Asynchronous flow control. When the $\overline{\text{CTS}}$ pin is negated, the transmitter stops at the end of the current character. If $\overline{\text{CTS}}$ is negated past the middle of the current character, the next full character can be sent and transmission is stopped. When $\overline{\text{CTS}}$ is asserted once more, transmission continues where it left off and no $\overline{\text{CTS}}$ lost error is reported. No characters except idles are transmitted while $\overline{\text{CTS}}$ is negated.

SL—Stop Length

This bit selects the number of stop bits transmitted by the SCC2 UART controller. It can be modified on-the-fly. The receiver is always enabled for one stop bit unless the SCC2 UART controller is in synchronous mode and the RZS bit is set. Fractional stop bits are configured in the general DSR, which is described in **Section 16.9.4 Data Synchronization Register**.

0 = One stop bit.
1 = Two stop bits.

CL—Character Length

This field determines the number of data bits in the character, not including the optional parity or multidrop address bits. When you use a character that is less than 8 bits, the most-signifiant bits in memory are written as zeros and they are labeled “don’t care” when they are transmitted. This field can be modified on-the-fly.

00 = 5 data bits.
01 = 6 data bits.
10 = 7 data bits.
11 = 8 data bits.

UM—UART Mode

This field selects the protocol that is implemented over the ASYNC channel and it can be modified on-the-fly.

00 = Normal UART operation. Multidrop mode is disabled and an idle-line wake-up is selected. In the idle-line wake-up mode, the UART receiver is reenabled by receiving a character of all ones.
01 = Manual multidrop mode. In multidrop mode, an additional address/data bit is transmitted with each character. The multidrop asynchronous modes are compatible with the MC68681 DUART, MC68HC11 SCI, DSP56000 SCI, and Intel 8051 serial interface. The UART receiver is reenabled when the last bit received in the character is a one. This means that the received character is an address that has to be processed by all inactive processors. The SCC2 UART controller receives the address character and writes it to a new buffer. The core then compares the written address with its own address and decides whether to ignore or process the following characters.
10 = Reserved.
11 = Automatic multidrop mode. In this mode, the communication processor module automatically checks the address of the incoming address character using the UADDR1 and UADDR2 parameter RAM values and accepts or discards the data that follows the address.

FRZ—Freeze Transmission

This bit allows you to stop the UART transmitter and continue transmission from the same point at a later time.

0 = Normal operation. If the SCC2 UART controller was previously frozen, the UART resumes transmission from the next character in the same buffer that was frozen.
1 = The SCC2 UART controller completes transmission of any data already transferred to the UART FIFO (the number of characters depends on the TFL bit in the GSMR_H) and then freezes. After this bit is reset, transmission proceeds from the next character.

RZS—Receive Zero Stop Bits

This bit configures the UART receiver to receive data without any stop bits. This configuration is useful in V.14 applications where SCC2 UART controller data is supplied synchronously and all stop bits of a particular character can be omitted for the purpose of cross-network rate adaptation. RZS should only be set if the SYN bit is also set.

0 = The receiver operates normally, but at least one stop bit is required between characters. A framing error is issued when there is a missing stop bit and a break status is set if a character with all-zero data bits is received with a zero stop bit.
1 = The receiver continues if a missing stop bit is detected. If the stop bit is a zero, the next bit is considered the first data bit of the next character. A framing error is issued if a stop bit is missing, but a break status is only reported after back-to-back reception of two break characters without stop bits.

SYN—Synchronous Mode

0 = Normal asynchronous operation. Normally, you program the TENC and RENC fields in the GSMR_L to NRZ and select either 8×, 16×, or 32× in the RDCR and TDCR fields of the GSMR_L. 16× is the recommended value for most applications.

1 = Synchronous SCC2 UART controller using 1× clock. Normally, you program the TENC and RENC fields in the GSMR_L to NRZ and set the RDCR and TDCR fields in the GSMR_L to 1× mode. A 1 bit is transferred with each clock and is synchronous to the clock. As with the other modes, the clock can be provided internally or externally. This mode is sometimes referred to as isochronous UART channel operation.

DRT—Disable Receiver While Transmitting

0 = Normal operation.

1 = While the SCC2 UART controller is transmitting data, the receiver is disabled. This is useful if the SCC2 UART controller is configured onto a multidrop line and you do not want to receive your transmission.

Note: You should set the preamble bit in the transmit buffer descriptor if you are using the MPC823 in multidrop UART mode.

Bit 10—Reserved

This bit is reserved and should be set to 0.

PEN—Parity Enable

0 = No parity.

1 = Parity is enabled and determined by the parity mode bits.

RPM—Receiver Parity Mode

This field selects the type of parity check that the receiver will perform. It can be modified on-the-fly.

00 = Odd parity.

01 = Low parity. Always check for a zero in the parity bit position.

10 = Even parity.

11 = High parity. Always check for a 1 in the parity bit position.

When odd parity is selected, the transmitter counts the number of ones in the data word. If the total number of ones is not an odd number, the parity bit is set to 1 and produces an odd number. If the receiver counts an even number of ones, an error in transmission has occurred. In the same manner, for even parity, an even number must result from the calculation performed at both ends of the line. In high or low parity (also called mark or space parity), if the parity bit is not high or low, a parity error is reported.

Note: The receive parity errors can be ignored, but not disabled.

TPM—Transmitter Parity Mode

This field selects the type of parity that the transmitter performs. It can be modified on-the-fly.

00 = Odd parity.
01 = Force low parity. Always send a zero in the parity bit position.
10 = Even parity.
11 = Force high parity. Always send a 1 in the parity bit position.

16.9.15.16 SCC2 UART RECEIVE BUFFER DESCRIPTORS. On a per-buffer basis, the communication processor module uses the receive (RX) buffer descriptors to report information about the received data. The communication processor module closes the current buffer, generates a maskable interrupt, and starts receiving data into the next buffer after one of the following events occur:

- A user-defined control character is received.
- An error occurs during message processing.
- A full receive buffer is detected.
- A MAX_IDL number of consecutive idle characters is received.
- The **ENTER HUNT MODE** command is issued.
- The **CLOSE RX BD** command is issued.

An address character is received in multidrop mode. The address character is written to the next buffer for a software comparison.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

An example of the RX buffer descriptor process is illustrated in Figure 16-74.

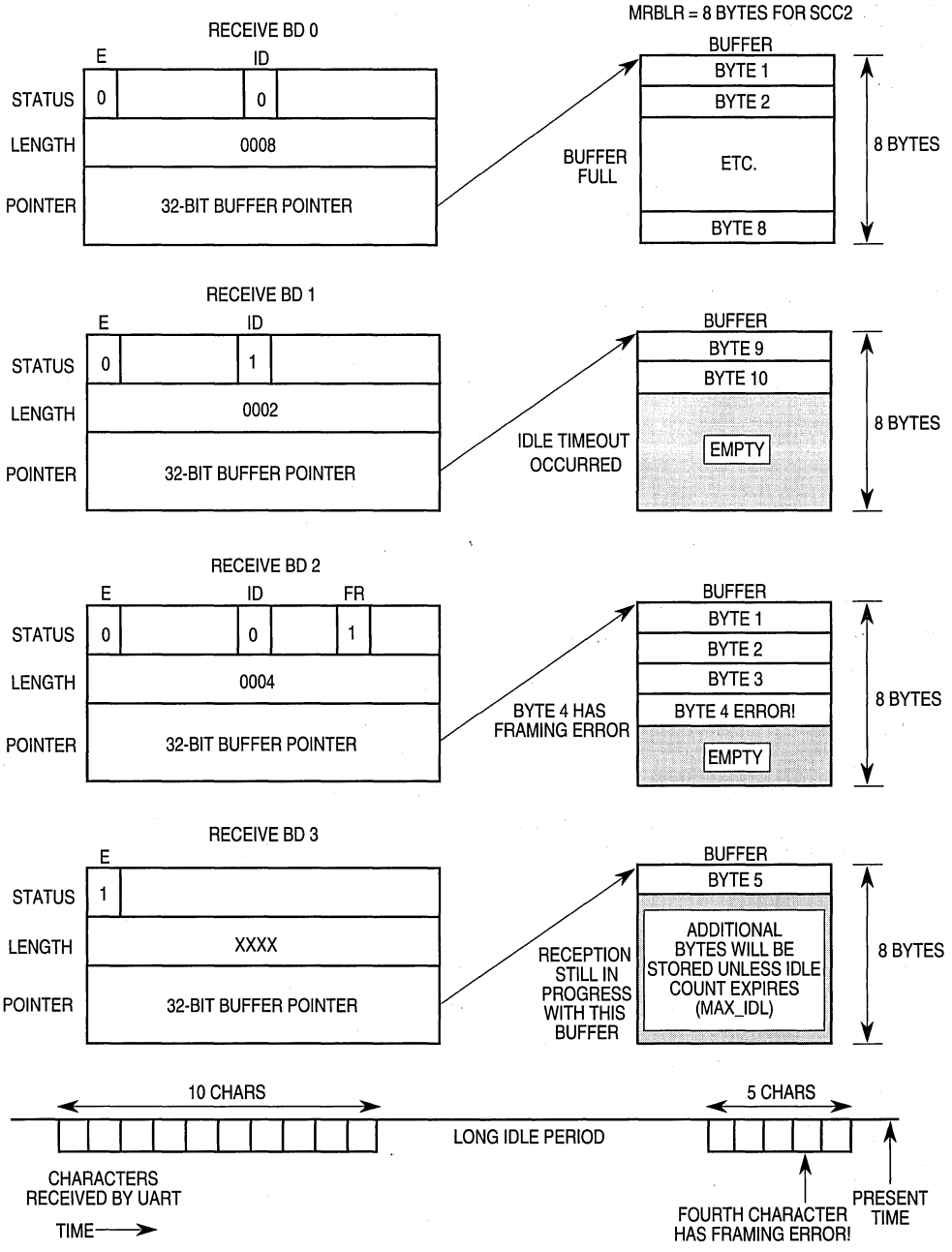


Figure 16-74. SCC2 UART Receive Buffer Descriptor Example

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	**E**	RES	**W**	**I**	C	**A**	**CM**	ID	AM	RES	BR	FR	PR	RES	OV	CD
OFFSET + 2	DATA LENGTH															
OFFSET + 4	RX DATA BUFFER POINTER															
OFFSET + 6																

NOTE: You are only responsible for initializing the items in bold.

E—Empty

0 = The data buffer associated with this RX buffer descriptor has been filled with data or reception has been aborted because of an error condition. The core is free to examine or write to any fields of this RX buffer descriptor. The communication processor module does not use this buffer descriptor again as long as the E bit is zero.

1 = The data buffer associated with this buffer descriptor is empty or is currently receiving data. This RX buffer descriptor and its associated receive buffer are owned by the communication processor module. Once the E bit is set, the core should not write any fields of this RX buffer descriptor.

Bits 1, 9, and 13—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the RX buffer descriptor table.

1 = This is the last buffer descriptor in the RX buffer descriptor table. After this buffer has been used, the communication processor module receives incoming data into the first buffer descriptor that RBASE points to in the table. The number of RX buffer descriptors in this table are programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is filled.

1 = The RX bit in the SCCE–UART register is set when this buffer is completely filled by the communication processor module, indicating the need for the core to process the buffer. The RX bit can cause an interrupt if it is enabled.

C—Control Character

0 = This buffer does not contain a control character.

1 = This buffer contains a control character. The last byte in the buffer is one of the user-defined control characters.

A—Address

0 = The buffer only contains data.
1 = When operating in manual multidrop mode, this bit indicates that the first byte of this buffer contains an address byte. The address comparison should be implemented in the software. In automatic multidrop mode, this bit indicates that the buffer descriptor contains a message that was received immediately after an address was recognized in UADDR1 or UADDR2. This address is not written into the receive buffer.

CM—Continuous Mode

0 = Normal operation.
1 = The E bit is not cleared by the communication processor module after this buffer descriptor is closed, thus allowing the associated data buffer to be automatically overwritten next time the communication processor module accesses this buffer descriptor. However, the E bit is cleared if an error occurs during reception, regardless of how the CM bit is set.

ID—Buffer Closed on Reception of Idles

This bit indicates that the buffer is closed because a programmable number of consecutive idle sequences (MAX_IDL) was received.

AM—Address Match

This bit is only significant if the address bit is set and the automatic multidrop mode is selected in the UM bits. After an address match, this bit defines which address character matched the address character that you defined, which enables the SCC2 UART controller to receive data.

0 = The address matched the value in UADDR2.
1 = The address matched the value in UADDR1.

BR—Break Received

This bit indicates that a break sequence has been received at the same time that data is being received into this buffer.

FR—Framing Error

This bit indicates that a character with a framing error has been received and located in the last byte of this buffer. A framing error is a character without a stop bit. A new receive buffer is used to receive more data.

PR—Parity Error

This bit indicates that a character with a parity error has been received and located in the last byte of this buffer. A new receive buffer is used to receive more data.

OV—Overrun

This bit indicates that a receiver overrun has occurred while the SCC2 UART controller was receiving a message.

CD—Carrier Detect Lost

This bit indicates that the carrier detect signal has been negated while the SCC2 UART controller was receiving a message.

DATA LENGTH

This field represents the number of octets that the communication processor module writes into this buffer descriptor's data buffer. It is written by the communication processor module once the buffer descriptor is closed.

Note: The actual amount of memory allocated for this buffer should be at least as much as the contents of the MRBLR.

RX DATA BUFFER POINTER

This field always points to the first location of the associated data buffer and they can be even or odd. The buffer can reside in internal or external memory.

16.9.15.17 SCC2 UART TRANSMIT BUFFER DESCRIPTOR. The communication processor module receives data transmitted on an SCC2 channel by arranging it in buffers referenced by the channel's transmit (TX) buffer descriptor table. Using the buffer descriptors, the communication processor module confirms transmission and indicates error conditions so that the processor knows that the buffers have been serviced.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	**R**	RES	**W**	**I**	**CR**	**A**	**CM**	P	NS	RESERVED						CT
OFFSET + 2	**DATA LENGTH**															
OFFSET + 4	**TX DATA BUFFER POINTER**															
OFFSET + 6																

NOTE: You are only responsible for initializing the items in bold.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

R—Ready

0 = The data buffer associated with this buffer descriptor is not ready to be transmitted. You are free to manipulate this buffer descriptor or its associated data buffer. The communication processor module clears this bit after the buffer is transmitted or after an error condition is encountered.

1 = The data buffer, which you must prepare for transmission, has not been transmitted yet or is currently being transmitted. You cannot write any fields of this buffer descriptor once this bit is set.

Bits 1 and 9–14—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the TX buffer descriptor table.

1 = This is the last buffer descriptor in the TX buffer descriptor table. After this buffer has been used, the communication processor module will transmit data from the first buffer descriptor that TBASE points to in the table. The number of TX buffer descriptors in this table are programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is serviced.

1 = The TX bit in the SCCE–UART register is set when this buffer is serviced by the communication processor module, which can cause an interrupt.

CR—Clear-to-Send Report

This bit allows you to choose between either no delay between buffers transmitted in SCC2 UART mode, or a more accurate $\overline{CTS}$ lost error report and three bits of idle between buffers.

0 = The buffer following this buffer is transmitted with no delay (assuming it is ready), but the CT bit may not be set in the correct TX buffer descriptor or may not be set at all in a $\overline{CTS}$ lost condition. Asynchronous flow control, however, continues to function normally.

1 = Normal $\overline{CTS}$ lost error reporting and three bits of idle occur between back-to-back buffers.

A—Address

This bit is only valid in multidrop mode. Either automatic or manual.

0 = This buffer only contains data.

1 = Set by the core, this bit indicates that this buffer contains address characters. All of the buffer data is transmitted as address characters.

CM—Continuous Mode

0 = Normal operation.
1 = The communication processor module does not clear the R bit after this buffer descriptor is closed, thus allowing the associated data buffer to be automatically retransmitted next time the communication processor module accesses this buffer descriptor. However, the R bit is cleared if an error occurs during transmission, regardless of how the CM bit is set.

P—Preamble

0 = No preamble sequence is sent.
1 = The SCC2 UART controller sends one character that consists of all ones before it sends the data so that the other remote receiver can detect an idle line before the data. If this bit is set and the data length of this buffer descriptor is zero, only a preamble is sent.

NS—No Stop Bit Transmitted

0 = Normal operation. Stop bits are sent with all characters in this buffer.
1 = By setting the SYN bit in the PSMR–SCC2 UART register, the data in this buffer is sent without stop bits if the Sync mode is selected. If Async is selected, the stop bit is transmitted as defined by the FSB field in the general DSR, as described in **Section 16.9.4 Data Synchronization Register**.

CT—$\overline{\text{CTS}}$ Lost

The communication processor module writes this bit after it finishes transmitting the associated data buffer.

0 = The $\overline{\text{CTS}}$ signal remains asserted during transmission.
1 = The $\overline{\text{CTS}}$ signal is negated during transmission.

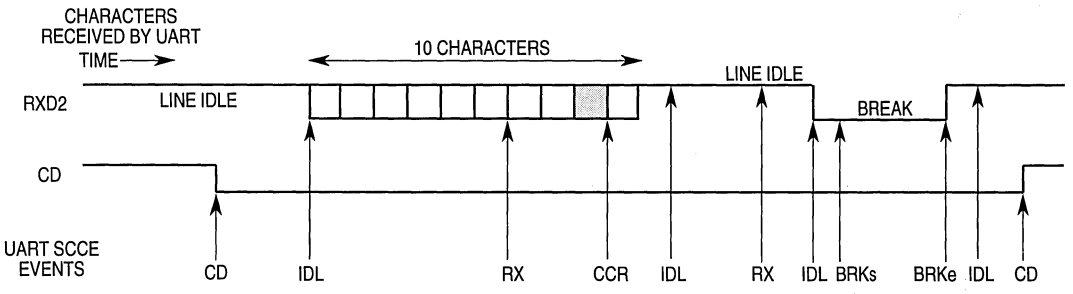
DATA LENGTH

This field represents the number of octets that the communication processor module should transmit from this buffer descriptor data buffer. It is never modified by the communication processor module. Normally, this value should be greater than zero. The data length can be equal to zero with the P bit set and only a preamble is sent. The communication processor module writes these bits after it finishes transmitting the associated data buffer.

TX DATA BUFFER POINTER

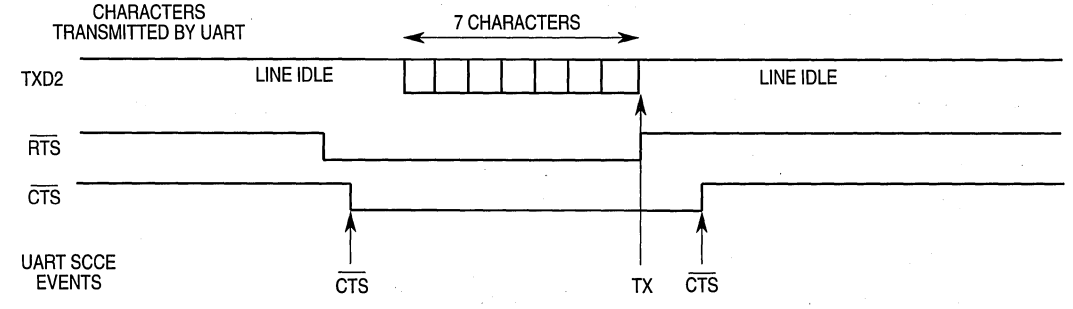
This field always points to the first location of the associated data buffer and can be even or odd. The buffer can reside in internal or external memory. The communication processor module writes these bits after it finishes transmitting the associated data buffer.

16.9.15.18 SCC2 UART EVENT REGISTER. When the SCC2 is in UART mode, the 16-bit memory-mapped SCC2 event register is referred to as the SCC2 UART event register (SCCE–UART). Since each protocol has specific requirements, the SCCE bits are different for each implementation. This register is used to report events recognized by the UART channel and to generate interrupts. When an event is recognized, the SCC2 UART controller sets the corresponding bit in the UART event register. Interrupts generated by this register can be masked in the SCCM–UART register. An example of interrupts that can be generated by the SCC2 UART controller is illustrated in Figure 16-75.



NOTES:
1. The first RX event assumes receive buffers are 6 bytes each.
2. The second IDL event occurs after an all-ones character is received.
3. The second RX event position is programmable based on the MAX_IDL value.
4. The BRKs event occurs after the first break character is received.
5. The CD event must be programmed in the port C parallel I/O, not in SCC2 itself.

LEGEND:
A receive control character defined not to be stored in the receive buffer.



NOTES:
1. TX event assumes all seven characters were put into a single buffer and CR = 1 in the Tx BD.
2. The CTS event must be programmed in the port C parallel I/O, not in the SCC itself.

Figure 16-75. SCC2 UART Interrupt Event Example

A bit is cleared by writing a 1 (writing a zero has no effect) and more than one bit can be cleared at a time. All unmasked bits must be cleared before the communication processor module clears the internal interrupt request. This register is cleared at reset and can be read at any time.

SCCE–UART

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED			GLR	GLT	RES	AB	IDL	GRA	BRKE	BRKS	RES	CCR	BSY	TX	RX
RESET	0			0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA30															

Bits 0–2, 5, and 11—Reserved

These bits are reserved and should be set to 0.

GLR—Glitch on RX

If set, this bit indicates that the SCC2 has encountered a clock glitch on the receive clock.

GLR—Glitch on TX

If set, this bit indicates that the SCC2 has encountered a clock glitch on the transmit clock.

AB—Auto Baud

If set, this bit indicates that an auto baud lock has been detected. The core should rewrite the baud rate generator with the precise divider value for the preferred baud rate. Refer to **Section 16.8 The Baud Rate Generators** for more details.

IDL—Idle Sequence Status Changed

If set, this bit indicates that a change in the serial line's status has been encountered on the UART channel. The real-time status of the line can be read in the SCCS–UART. Idle is entered when a character of all ones is received and it is exited when a single zero is received.

GRA—Graceful Stop Complete

If set, this bit indicates that a graceful stop, which is initiated by the **GRACEFUL STOP TRANSMIT** command, is complete. This bit is set as soon as the transmitter has finished with any buffer in progress when the command was issued. It is set immediately if no buffer was in progress when the command was issued.

BRKE—Break End

If set, this bit indicates that an end of a break sequence has been found. This indication is set no sooner than after an idle bit is received following a break sequence.

BRKS—Break Start

If set, this bit indicates when a break character is received. It is the first break of a break sequence. You do not receive multiple BRKS events if a long break sequence is received.

CCR—Control Character Received

If set, this bit indicates when a control character is received and stored in the receive control character register (RCCRP).

BSY—Busy Condition

If set, this bit indicates that a character has been received and discarded due to a lack of buffers. If multidrop mode is selected, the receiver automatically enters hunt mode. Otherwise, reception continues as soon as an empty buffer is provided. The latest point that an RX buffer descriptor can be changed to empty and still guarantee avoiding the busy condition, is the middle of the stop bit of the first character to be stored in that buffer.

TX—TX Buffer

If set, this bit indicates that a buffer has been transmitted over the UART channel. If CR = 1 in the TX buffer descriptor, this bit is set no sooner than when the last stop bit of the last character in the buffer is first transmitted. If CR = 0, this bit is set after the last character is written to the transmit FIFO.

RX—RX Buffer

If set, this bit indicates that a buffer has been received over the UART channel. This event occurs no sooner than the middle of the first stop bit of the character that caused the buffer to close.

16.9.15.19 SCC2 UART MASK REGISTER. When the SCC2 is in UART mode, the 16-bit, read/write SCC2 mask register is referred to as the SCC2 UART mask register (SCCM–UART). It has the same bit formats as the SCCE–UART register. If a bit in this register is a 1, the corresponding interrupt in the SCCE–UART register is enabled. If it is zero, the corresponding interrupt in the event register is masked.

SCCM–UART

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED			GLR	GLT	RES	AB	IDL	GRA	BRKE	BRKS	RES	CCR	BSY	TX	RX
RESET	0			0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA34															

16.9.15.20 SCC2 UART STATUS REGISTER. When the SCC2 is in UART mode, the 8-bit read-only SCC2 status register is referred to as the SCC2 UART status register (SCCS–UART). Since each protocol has specific requirements, the SCCS bits are different for each implementation. This register allows you to monitor real-time status conditions on the RXD2 pin. The real-time status of the $\overline{CTS}$ and $\overline{CD}$ pins are part of the port C parallel I/O.

SCCS–UART

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED							ID
RESET	0							0
R/W	R							R
ADDR	(IMMR & 0xFFFF0000) + 0xA37							

Bits 0–6—Reserved

These bits are reserved and should be set to 0.

ID—Idle Status

This bit is set when the RXD2 pin has been a logic one for at least a full character time.

0 = The pin is not idle.
1 = The pin is idle.

16.9.15.21 SCC2 UART PROGRAMMING EXAMPLE. The following initialization sequence is for the 9,600 baud, 8 data bits, no parity, and stop bit of an SCC2 in UART mode assuming a 25MHz system frequency. BRG1 is used in the example. The SCC2 UART controller is configured with the $\overline{RTS2}$, $\overline{CTS2}$, and $\overline{CD2}$ pins active and the $\overline{CTS2}$ pin is used as an automatic flow control signal.

1. Configure the port A pins to enable the TXD2 and RXD2 pins. Write PAPAR bits 13 and 12 with ones and then write the PADIR and PAODR bits 13 and 12 with zeros.
2. Configure the port C pins to enable $\overline{RTS2}$, $\overline{CTS2}$, and $\overline{CD2}$. Write PCPAR bit 14 with one and bits 9 and 8 with zeros, PCDIR bits 14, 9, and 8 with zeros, and PCSO bits 9 and 4 with ones.
3. Configure BRG1, and write BRGC1 with 0x010144. The DIV16 bit is not used and the divider is 162 (decimal). The resulting BRG1 clock is 16× the preferred bit rate of the SCC2 in UART mode.
4. Connect the BRG1 clock to the SCC2 using the serial interface. Write the R2CS field in the SICR to 000 and the T2CS field in the SICR to 000.
5. Write 0x0001 to the SDCR to set the SDMA bus arbitration level to 5.
6. Connect the SCC2 to the NMSI and clear the SC2 bit in the SICR.

7. Write to RBASE and TBASE in the SCC2 parameter RAM to point to the RX and TX buffer descriptors in the dual-port RAM. Assuming one RX buffer descriptor at the beginning of dual-port RAM and one TX buffer descriptor following that RX buffer descriptor, write RBASE with 0x2000 and TBASE with 0x2008.
8. Write 0x0041 to the CPCR to execute the **INIT RX AND TX PARAMS** command for SCC2.
9. Write 0x15 into the RFCR and 0x15 into the TFCR for normal operation.
10. Write MRBLR with the maximum number of bytes per receive buffer. For this case, assume 16 bytes, so MRBLR = 0x0010.
11. Write MAX_IDL with 0x0000 in the SCC2 UART parameter RAM to disable the MAX_IDL functionality for this example.
12. Write 0x0001 to the BRKCR, so that if a **STOP TRANSMIT** command is issued, one break character is sent.
13. Clear PAREC, FRMEC, NOSEC, and BRKEC in the SCC2 UART parameter RAM.
14. Clear UADDR1 and UADDR2. They are not used.
15. Clear TOSEQ. It is not used.
16. Write CHARACTER1–CHARACTER8 with 0x8000. They are not used.
17. Write RCCM with 0xC0FF. It is not used.
18. Initialize the RX buffer descriptor. Assume the RX data buffer is at 0x00001000 in main memory. Write 0xB000 to RX_BD_Status, 0x0000 to RX_BD_Length (optional), and 0x00001000 to RX_BD_Pointer.
19. Initialize the TX buffer descriptor. Assume the TX data buffer is at 0x00002000 in main memory and contains five 8-bit characters. Write 0xB000 to TX_BD_Status, 0x0010 to TX_BD_Length, and 0x00002000 to TX_BD_Pointer.
20. Write 0xFFFF to the SCCE–UART register to clear any previous events.
21. Write 0x0003 to the SCCM–UART register to enable the transmit and receive interrupts.
22. Write 0x20000000 to the CIMR to allow SCC2 to generate a system interrupt. The CICR should also be initialized.
23. Write 0x00000020 to the GSMR_H to configure a small receive FIFO width.
24. Write 0x00028004 to the GSMR_L to configure 16× oversampling for transmit and receive, the $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ pins to automatically control transmission and reception (DIAG field) and the SCC2 UART mode. Notice that the transmitter (ENT bit) and receiver (ENR bit) have not been enabled yet.
25. Set the PSMR–SCC2 UART to 0xB000 to configure automatic flow control using the $\overline{\text{CTS}}$ pin, 8-bit characters, no parity, 1 stop bit, and asynchronous SCC2 UART operation.
26. Write 0x00028034 to the GSMR_L register to enable the SCC2 transmitter and receiver. This additional write ensures that the ENT and ENR bits are enabled last.

Note: After 16 bytes are transmitted, the TX buffer descriptor is closed. Additionally, the receive buffer is closed after 16 bytes are received. Any data received after 16 bytes causes a busy (out-of-buffers) condition since only one RX buffer descriptor is prepared.

16.9.15.22 S-RECORD PROGRAMMING EXAMPLE. The following is an example of a downloading application that uses the SCC2 channel as a UART controller. The application performs S-record downloads and uploads between a host computer and an intelligent peripheral through a serial asynchronous line. The S-records are strings of ASCII characters that begin with 'S' and end in an end-of-line character. This characteristic is used to impose a message structure on the communication between the devices. Each device can transmit XON and XOFF characters for flow control, which do not form part of the program being uploaded or downloaded.

The PSMR—SCC2 UART register should be set as needed with the FRZ bit cleared and the ENT and ENR bits set. Receive buffers should be linked to the receive buffer table with the I bit set. For simplicity, assume that the line is not multidrop (no addresses are transmitted) and that each S-record fits into a single data buffer. Three characters should first be entered into the SCC2 UART control character table:

- Line Feed—Both the E and R bits should be cleared. When an end-of-line character is received, the current buffer is closed and made available to the core for processing. This buffer contains an entire S record that the processor can now check and copy to memory or disk as required.
- XOFF—E should be cleared and R should be set. Whenever the core receives a control character received interrupt and the receive control character register contains XOFF, the software should immediately stop transmitting to the other station by setting the FRZ bit in the PSMR–SCC2 UART. This prevents data from being lost by the other station when it runs out of receive buffers.
- XON—This character should be received after XOFF. The E bit should be cleared and the R bit should be set. The FRZ bit on the transmitter should now be cleared. The communication processor module automatically resumes transmission of the serial line at the point at which it was previously stopped. Like XOFF, the XON character is not stored in the receive buffer.

To receive the S-records, the core must only wait for the receive interrupt, thus indicating that a complete S-record buffer has been received. Transmission requires assembling S-records into data buffers and linking them to the transmit buffer table and it can be temporarily stopped when an XOFF character is received. This scheme minimizes the number of interrupts received by the core (one per S-record) and relieves it from the task of continually scanning for control characters.

16.9.16 The SCC2 In HDLC Mode

Layer 2 of the seven-layer open systems interconnection (OSI) model from ISO is the data link layer and one of the most common protocols in this layer is high-level data link control (HDLC). In fact, many other common layer 2 protocols—SDLC, SS#7, AppleTalk, LAPB, and LAPD—are based on HDLC and its framing structure in particular. The framing structure of HDLC is illustrated in Figure 16-76.

HDLC uses a zero insertion/deletion process (commonly known as bit-stuffing) to ensure that the bit pattern of the delimiter flag does not occur in the fields between flags. The HDLC frame is synchronous and relies on the physical layer to provide a method for clocking and synchronizing the transmitter/receiver.

Since the layer 2 frame can be transmitted over a point-to-point link, a broadcast network, or packet and circuit switched systems, an address field is needed to carry the frame's destination address. The length of this field is commonly 0, 8, or 16 bits, depending on the data link layer protocol. For instance, SDLC and LAPB use an 8-bit address and SS#7 has no address field at all because it is always used in point-to-point signaling links. LAPD further divides its 16-bit address into different fields to specify various access points within one piece of equipment. It also defines a broadcast address. Some HDLC-type protocols allow for extended addressing beyond 16 bits.

The 8- or 16-bit control field provides a flow control number and defines the frame type (control or data). The exact use and structure of this field depends on the protocol using the frame. Data is transmitted in the data field and its length is dependent on the protocol of the frame. Layer 3 frames are carried in this data field. Error control is implemented by appending a cyclic redundancy check (CRC) to the frame, which in most protocols is 16 bits long but can be as long as 32 bits. In HDLC, the least-significant bit of each octet is transmitted first and the most-significant bit of the CRC is transmitted first.

When the MODE field of the GSMR_L selects the HDLC mode, the serial communication controller is functioning in HDLC mode. When you use an SCC2 in HDLC mode with a nonmultiplexed modem interface, the serial communication controller outputs are connected directly to the external pins. Modem signals can be supported through the port B and C pins. The receive and transmit clocks can be supplied from either the bank of baud rate generators, by the DPLL, or externally. You can also connect the SCC2 in HDLC mode to the TDM channel of the serial interface and use it with the time-slot assigner. The SCC2 in HDLC mode, also called the SCC2 HDLC controller, consists of separate transmit and receive sections whose operations are asynchronous with the core. You can allocate up to 196 buffer descriptors so that you can transmit or receive many frames without any interference from the host.

16.9.16.1 FEATURES. The following list summarizes the main features of the SCC2 in HDLC mode:

- Flexible data buffers with multiple buffers per frame
- Separate interrupts for frames and buffers (receive and transmit)
- Received frames threshold to reduce interrupt overhead
- Can be used with the SCC2 DPLL
- Four address comparison registers with mask
- Maintenance of five 16-bit error counters
- Flag/abort/idle generation or detection
- Zero insertion/deletion
- 16- or 32-bit CRC-CCITT generation/checking
- Detection of nonoctet aligned frames
- Detection of frames that are too long
- Programmable flags (0–15) between successive frames
- Automatic retransmission in case of collision

16.9.16.2 SCC2 HDLC CHANNEL FRAME TRANSMISSION PROCESS. The HDLC transmitter is designed to work with little or no intervention from the core. When the core enables one of the transmitters, it starts transmitting flags or idles as programmed in the PSMR–SCC2 HDLC register. The SCC2 HDLC controller polls the first buffer descriptor in the transmit channel buffer descriptor table. When there is a frame to transmit, the SCC2 HDLC controller fetches the data from memory and starts transmitting the frame after it transmits the minimum number of flags that you specify between frames. When the end of the current buffer descriptor has been reached and the last buffer in the frame bit is set, the CRC and closing flag are appended. In HDLC mode, the least-significant bit of each octet and the most-significant bit of the CRC are transmitted first. An HDLC frame is illustrated in Figure 16-76.

OPENING FLAG	ADDRESS	CONTROL	INFORMATION (OPTIONAL)	CRC	CLOSING FLAG
8 BITS	16 BITS	8 BITS	8N BITS	16 BITS	8 BITS

Figure 16-76. SCC2 HDLC Framing Structure

After a closing flag is transmitted, the SCC2 HDLC controller writes the frame status bits into the buffer descriptor and clears the R bit. When you reach the end of the current buffer descriptor and the last bit is not set, only the R bit is cleared. In either mode, an interrupt can be issued if the I bit in the TX buffer descriptor is set. The SCC2 HDLC controller then proceeds to the next TX buffer descriptor in the table. This method allows you to be interrupted after each buffer, a specific buffer, or each frame.

To rearrange the transmit queue before the communication processor module finishes transmitting all of the buffers, issue the **STOP TRANSMIT** command. This can be useful for transmitting expedited data before previously linked buffers or when an error occurs. When receiving the **STOP TRANSMIT** command, the SCC2 HDLC controller stops transmitting the current frame and starts transmitting idles or flags. When the SCC2 HDLC controller receives the **RESTART TRANSMIT** command, it resumes transmission. To insert a high-priority frame without aborting the current frame, the **GRACEFUL STOP TRANSMIT** command can be issued. A special interrupt can be generated in the event register when the current frame is complete.

16.9.16.3 SCC2 HDLC CHANNEL FRAME RECEPTION PROCESS. The HDLC receiver is designed to work with little or no intervention from the core and can perform address recognition, CRC checking, and maximum frame length checking. You are free to use the received frame to perform any HDLC-based protocol.

When the core enables one of the receivers, the receiver waits for an opening flag character and when it detects the first byte of the frame, the SCC2 HDLC controller compares the frame address against the user-programmable addresses. You have four 16-bit address registers and an address mask available for address matching. The SCC2 HDLC controller compares the received address field to the user-defined values after masking with the address mask. The SCC2 HDLC controller can also detect broadcast (all ones) address frames if one address register is written with all ones.

If a match is detected, the SCC2 HDLC controller fetches the next buffer descriptor and if it is empty, it starts transferring the incoming frame to the buffer descriptor associated data buffer. When the data buffer has been filled, the SCC2 HDLC controller clears the E bit in the buffer descriptor and generates an interrupt if the I bit in the buffer descriptor is set. If the incoming frame exceeds the length of the data buffer, the SCC2 HDLC controller fetches the next buffer descriptor in the table and, if it is empty, continues transferring the rest of the frame to this buffer descriptor associated data buffer.

During this process, the SCC2 HDLC controller checks for a frame that is too long. When the frame ends, the CRC field is checked against the recalculated value and written to the data buffer. The data length written to the last buffer descriptor in the HDLC frame is the length of the entire frame. This enables HDLC protocols that "lose" frames to correctly recognize the frame-too-long condition. The SCC2 HDLC controller then sets the last buffer in frame bit, writes the frame status bits into the buffer descriptor, and clears the E bit. The SCC2 HDLC controller next generates a maskable interrupt, indicating that a frame has been received and is in memory. The SCC2 HDLC controller then waits for a new frame. Back-to-back frames can be received with only a single shared flag between frames.

In the received frames threshold (RFTHR) location of the parameter RAM you can configure the SCC2 HDLC controller not to interrupt the core until a certain number of frames are received. You can combine this function with a timer to implement a timeout if less than the threshold number of frames are received.

Note: The SCC2 HDLC controller must receive a maximum of eight clocks (after a frame is received) to complete the reception.

16.9.16.4 SCC2 HDLC PARAMETER RAM MEMORY MAP. When configured to operate in HDLC mode, the SCC2 overlays the structure used in Table 16-24 with the HDLC parameters that are described in Table 16-27 below.

Table 16-27. SCC2 HDLC Parameter RAM Memory Map

ADDRESS	NAME	WIDTH	DESCRIPTION
SCC2 Base + 30	RES	Word	Reserved
SCC2 Base + 34	**C_MASK**	Word	CRC Constant
SCC2 Base + 38	**C_PRES**	Word	CRC Preset
SCC2 Base + 3C	**DISFC**	Half-word	Discard Frame Counter
SCC2 Base + 3E	**CRCEC**	Half-word	CRC Error Counter
SCC2 Base + 40	**ABTSC**	Half-word	Abort Sequence Counter
SCC2 Base + 42	**NMARC**	Half-word	Nonmatching Address RX Counter
SCC2 Base + 44	**RETRC**	Half-word	Frame Transmission Counter
SCC2 Base + 46	**MFLR**	Half-word	Max Frame Length Register
SCC2 Base + 48	MAX_CNT	Half-word	Max_Length Counter
SCC2 Base + 4A	**RFTHR**	Half-word	Received Frames Threshold
SCC2 Base + 4C	RFCNT	Half-word	Received Frames Count
SCC2 Base + 4E	**HMASK**	Half-word	User-Defined Frame Address Mask
SCC2 Base + 50	**HADDR1**	Half-word	User-Defined Frame Address
SCC2 Base + 52	**HADDR2**	Half-word	User-Defined Frame Address
SCC2 Base + 54	**HADDR3**	Half-word	User-Defined Frame Address
SCC2 Base + 56	**HADDR4**	Half-word	User-Defined Frame Address
SCC2 Base + 58	TMP	Half-word	Temp Storage
SCC2 Base + 5A	TMP_MB	Half-word	Temp Storage

NOTE: You are only responsible for initializing the items in bold.
SCC2 base = (IMMR & 0xFFFF0000) + 0x3D00.
All references to registers in the parameter RAM table are actually implemented in the dual-port RAM area as a memory-based register.

- C_MASK—For the 16-bit CRC-CCITT, C_MASK should be initialized with 0x0000F0B8. For the 32-bit CRC-CCITT, C_MASK should be initialized with 0xDEBB20E3.
- C_PRES—For the 16-bit CRC-CCITT, C_PRES should be initialized with 0x0000FFFF. For the 32-bit CRC-CCITT, C_PRES should be initialized with 0xFFFFFFFF.
- DISFC, CRCEC, ABTSC, NMARC, and RETRC—These 16-bit (modulo 2^{16}) counters are maintained by the communication processor module. You can initialize them while the channel is disabled. The counters are as follows:
  - DISFC–Discarded Frame Counter (error-free frames, but no free buffers).
  - CRCEC–CRC Error Counter. Includes frames not addressed to you or frames received in the BSY condition, but does not include overrun errors.
  - ABTSC–Abort Sequence Counter.
  - NMARC–Nonmatching Address Received Counter (error-free frames only).
  - RETRC–Frame Retransmission Counter (due to collision).
- MFLR—The SCC2 HDLC controller compares the length of an incoming HDLC frame with the user-defined value given in this 16-bit register. If this limit is exceeded, the remainder of the incoming HDLC frame is discarded and the LG bit is set in the last RX buffer descriptor belonging to that frame. The SCC2 HDLC controller waits until the end of the frame and then reports the frame status and length in the last RX buffer descriptor. The MFLR is defined as all the in-frame bytes between the opening and closing flags.
- MAX_CNT—A temporary down-counter used to track the frame length.
- RFTHR—The received frame's threshold value is used to reduce the interrupt overhead that might otherwise occur when a series of short HDLC frames arrives, each causing an RXF interrupt. By setting the RFTHR value, you limit the frequency of RXF interrupts, which only occurs when the RFTHR value is reached.

Note: You should provide enough empty RX buffer descriptors to receive the number of frames specified in the RFTHR.

- RFCNT—A temporary down-counter uned to implement the RFTHR feature.
- HMASK, HADDR1, HADDR2, HADDR3, and HADDR4—The SCC2 HDLC controller has five 16-bit registers for address recognition: one mask register and four address registers. The SCC2 HDLC controller reads the frame address from the HDLC receiver, compares it against the four address register values, and then masks the result with the user-defined mask register. A one in the mask register represents a bit position for which address comparison should occur and a zero represents a masked bit position. When an address match is made, the address and the data following it are written into the data buffers. When the addresses are not matched and the frame is error-free, the nonmatching address received counter (NMARC) is incremented.

- TMP—A temporary register that is only used by the communication processor module.
- TMP_MB—A temporary register that is only used by the communication processor module.

Note: For 8-bit addresses, the eight high-order bits in HMASK should be masked out (cleared). The eight low-order bits of HMASK and HADDRx should contain the address byte that immediately follows the opening flag. For, example, to recognize a frame that begins 0x7E, 0x68, 0xAA, using 16-bit address recognition, HADDRx should contain 0xAA68 and HMASK should contain 0xFFFF. Refer to Figure 16-77 for details.

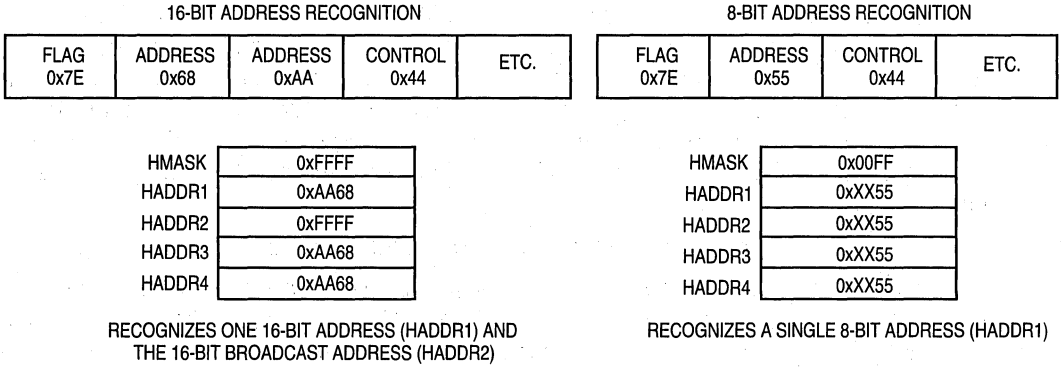


Figure 16-77. HDLC Address Recognition Example

16.9.16.5 PROGRAMMING THE SCC2 IN HDLC MODE. The core configures the serial communication controller to operate in one of the protocols set in the MODE field of the GSMR_L. The SCC2 HDLC controller uses the same data structure as other modes and it supports multibuffer operation and address comparisons. The reception errors are reported through the RX buffer descriptor and the transmit errors are reported through the TX buffer descriptor.

16.9.16.6 SCC2 HDLC COMMANDS. You can program the CPM command register (CPCR) with the following commands to transmit data.

- **STOP TRANSMIT**—After the hardware or software is reset and the channel is enabled in the PSMR–SCC2 HDLC register, the channel is in transmit enable mode and starts polling the first buffer descriptor in the table every 64 transmit clocks or immediately if the TOD bit is set in the transmit-on-demand register (TODR). The channel **STOP TRANSMIT** command disables the transmission of frames on the transmit channel. If the SCC2 HDLC controller receives this command while a frame is transmitting, transmission is aborted after a maximum of 64 additional bits and the transmit FIFO is flushed. The TBPTR is not advanced, no new buffer descriptor is accessed, and no new frames are transmitted for this channel. The transmitter transmits an abort sequence consisting of 01111111 (if the command was given during frame transmission) and begins transmitting flags or idles, as indicated by the PSMR–SCC2 HDLC register.

Note: If the MFF bit in the PSMR–SCC2 UART is set, one or more small frames can be flushed from the transmit FIFO. Issue the **GRACEFUL STOP TRANSMIT** command to stop this from occurring.

- **GRACEFUL STOP TRANSMIT**—This command is used to stop transmission smoothly, instead of abruptly, like the regular **STOP TRANSMIT** command. It stops transmission after the current frame is finished or immediately if there is no frame being transmitted. The GRA bit in the SCCE–HDLC is set once transmission has stopped. Then the HDLC transmit parameters and their buffer descriptors can be modified. The TBPTR points to the next TX buffer descriptor in the table. Transmission begins once the R bit of the next buffer descriptor is set and the **RESTART TRANSMIT** command is issued.
- **RESTART TRANSMIT**—This command enables characters to be transmitted on the transmit channel. The SCC2 HDLC controller expects this command after a **STOP TRANSMIT** command is issued or after a **GRACEFUL STOP TRANSMIT** command is issued or a transmitter error occurs. The SCC2 HDLC controller resumes transmission from the current TBPTR in the channel TX buffer descriptor table.
- **INIT TX PARAMETERS**—This command initializes all transmit parameters in this serial channel parameter RAM to their reset state and should only be issued when the transmitter is disabled. The **INIT TX AND RX PARAMS** command can be used to reset the transmit and receive parameters.

You can program the CPM command register with the following commands to receive data.

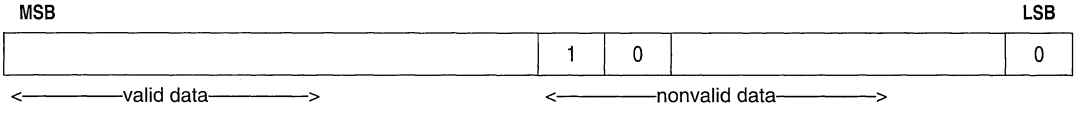
- **ENTER HUNT MODE**—After the hardware or software is reset and the channel is enabled in the PSMR–HDLC register, the channel is in receive enable mode and uses the first buffer descriptor in the table. The **ENTER HUNT MODE** command is generally used to force the HDLC receiver to stop receiving the current frame and enter hunt mode. In hunt mode, the SCC2 HDLC controller continually scans the input datastream for the flag sequence. After receiving the command, the current receive buffer is closed and the CRC is reset. Further frame reception uses the next buffer descriptor.
- **CLOSE RX BD**—This command should not be used in the HDLC protocol.
- **INIT RX PARAMETERS**—This command initializes all the receive parameters in this serial channel parameter RAM to their reset state and should only be issued when the receiver is disabled. The **INIT TX AND RX PARAMS** command can be used to reset the receive and transmit parameters.

16.9.16.7 SCC2 HDLC CONTROLLER ERRORS. The SCC2 HDLC controller reports frame reception and transmission error conditions using the channel buffer descriptors, error counters, and SCCE–HDLC register. The following transmission errors can be detected by the SCC2 HDLC controller.

- Transmitter Underrun Error—When this error occurs, the channel terminates buffer transmission, closes the buffer, sets the UN bit in the buffer descriptor, and generates the TXE interrupt if it is enabled. The channel continues transmitting after it receives the **RESTART TRANSMIT** command. The transmit FIFO size is 32 bytes for the serial communication controller.
- CTS Lost During Frame Transmission Error—When this error occurs, the channel terminates buffer transmission, closes the buffer, sets the CT bit in the buffer descriptor, and generates the TXE interrupt if it is enabled. The channel continues transmitting after it receives the **RESTART TRANSMIT** command. If this error occurs on the first or second buffer of the frame and the RTE bit in the PSMR–HDLC is set, the channel retransmits the frame when the $\overline{\text{CTS}}$ signal becomes active again. When you are working in SCC2 HDLC mode with the possibility of a collision, in order to ensure the retransmission method functions properly, the first and second data buffers should contain more than 36 bytes of data and 20 bytes of data if multiple buffers per frame are used. The channel also increments the retransmission counter. This requirement does not apply to small frames that consist of a single buffer.

The following reception errors can be detected by the SCC2 HDLC controller.

- Overrun Error—The SCC2 HDLC controller maintains an internal FIFO for receiving data. The communication processor module begins programming the SDMA channel and updating the CRC when 8 or 32 bits are received in the FIFO. When a receive FIFO overrun occurs, the channel writes the received data byte to the internal FIFO over the previously received byte. The previous data byte and the frame status are lost. The channel closes the buffer with the OV bit in the buffer descriptor set and generates the RXF interrupt if it is enabled. The receiver then enters hunt mode. Even if an overrun occurs during a frame whose address is not matched in the address recognition logic, an RX buffer descriptor with a data length of two is opened to report the overrun and the RXF interrupt is generated if it is enabled.
- CD Lost During Frame Reception Error—When this error occurs, the channel terminates frame reception, closes the buffer, sets the CD bit in the RX buffer descriptor, and generates the RXF interrupt if it is enabled. This error has the highest priority. The rest of the frame is lost and other errors are not checked in that frame. At this point, the receiver enters hunt mode.
- Abort Sequence Error—This error occurs when seven or more consecutive ones are received. When it does occur and the SCC2 HDLC controller receives a frame, the channel closes the buffer by setting the AB bit in the RX buffer descriptor and generating the RXF interrupt, if enabled. The channel also increments the abort sequence counter. The CRC and nonoctet error status conditions are not checked on aborted frames. The receiver then enters hunt mode. When an abort is received, you are given no indication that an SCC2 HDLC controller is not currently receiving a frame.
- Nonoctet Aligned Frame Error—When this error occurs, the channel writes the received data to the data buffer, closes the buffer, sets the NO bit in the RX buffer descriptor, and generates the RXF interrupt if it is enabled. The CRC error status should be disregarded on nonoctet frames. After a nonoctet aligned frame is received, the receiver enters hunt mode. An immediate back-to-back frame is still received. The nonoctet data may be derived from the last word in the data buffer as follows:



Note: If you are using the data buffer swapping option, the above diagram refers to the last byte of the data buffer, not the last word. In SCC2 HDLC mode, the LSB of each octet is transmitted first and the MSB of the CRC is transmitted first.

- CRC—When this error occurs, the channel writes the received CRC to the data buffer, closes the buffer, sets the CR bit in the RX buffer descriptor, and generates the RXF interrupt if it is enabled. The channel also increments the CRC error counter. After receiving a frame with a CRC error, the receiver enters hunt mode. An immediate back-to-back frame is still received. CRC checking cannot be disabled, but the CRC error can be ignored if checking is not required.

16.9.16.8 SCC2 HDLC MODE REGISTER. When the SCC2 is in HDLC mode, the 16-bit, memory-mapped, read/write protocol-specific mode register is referred to as the SCC2 HDLC mode register (PSMR–SCC2 HDLC). Since each protocol has specific requirements, the PSMR bits are different for each implementation.

PSMR–SCC2 HDLC

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	NOF				CRC		RTE	RES	FSE	DRT	BUS	BRM	MFF	RESERVED		
RESET	0				0		0	0	0	0	0	0	0	0		
R/W	R/W				R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W		
ADDR	(IMMR & 0xFFFF0000) + 0xA28															

NOF—Number of Flags

This field signifies the minimum number of flags between or before frames. If NOF = 0000, then no flags are inserted between the frames. Thus, the closing flag of one frame is immediately followed by the opening flag of the next frame in the case of back-to-back frames. These bits can be modified on-the-fly.

CRC—CRC Selection

00 = 16-bit CCITT-CRC (HDLC). X16 + X12 + X5 + 1.
01 = Reserved.
10 = 32-bit CCITT-CRC (Ethernet and HDLC). X32 + X26 + X23 + X22 + X16 + X12 + X11 + X10 + X8 + X7 + X5 + X4 + X2 + X1 +1.
11 = Reserved.

RTE—Retransmit Enable

0 = No retransmission.
1 = Automatic frame retransmission is enabled. Retransmission only occurs if the lost $\overline{\text{CTS}}$ occurs on the first or second buffer of the frame.

Bits 7 and 13–15—Reserved

These bits are reserved and should be set to 0.

FSE—Flag Sharing Enable.

This bit is only valid if the RTSM bit is set in the GSMR_H. This bit can be modified on-the-fly.

0 = Normal operation.
1 = If NOF0–NOF3 = 0000, then a single shared flag is transmitted between back-to-back frames. Other values of NOF0–NOF3 are decremented by 1 when this bit is set. This is useful in Signaling System #7 (SS#7) applications.

DRT—Disable Receiver While Transmitting

0 = Normal operation.
1 = While data is being transmitted by the serial communication controller, the receiver is disabled. This configuration is useful if the HDLC channel is configured onto a multidrop line and you do not want to receive your own transmission.

BUS—HDLC Bus Mode

0 = Normal HDLC operation.
1 = HDLC bus operation is selected. Refer to **Section 16.9.17 The HDLC Bus Controller** for more details.

BRM—HDLC Bus $\overline{\text{RTS}}$ Mode

This bit is only valid if BUS = 1. Otherwise, it is ignored.

0 = Normal $\overline{\text{RTS}}$ operation during HDLC bus mode. $\overline{\text{RTS}}$ is asserted on the first bit of the transmit frame and negated after the first collision bit is received.
1 = Special $\overline{\text{RTS}}$ operation during HDLC bus mode. $\overline{\text{RTS}}$ is delayed by one bit with respect to the normal case. This is useful when the HDLC bus protocol is being run locally and transmitted over a long-distance transmission line at the same time. Data can be delayed one bit before it is sent over the transmission line, thus $\overline{\text{RTS}}$ can be used to enable the transmission line buffers. The result is a clean signal level sent over the transmission line.

MFF—Multiple Frames in FIFO

0 = Normal operation. The transmit FIFO must never contain more than one HDLC frame. The $\overline{\text{CTS}}$ lost status is reported accurately on a per-frame basis. The receiver is not affected by this bit.
1 = The transmit FIFO can contain multiple frames, but lost $\overline{\text{CTS}}$ is not guaranteed to be reported on the exact buffer/frame it truly occurred on. This option, however, can improve the performance of HDLC transmissions of small back-to-back frames or in cases where you prefer to limit the number of flags transmitted between frames. The receiver is not affected by this bit.

16.9.16.9 SCC2 HDLC RECEIVE BUFFER DESCRIPTOR. The SCC2 HDLC controller uses the receive (RX) buffer descriptor to report information about the data received for each buffer. An example of the RX buffer descriptor process is illustrated in Figure 16-78.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

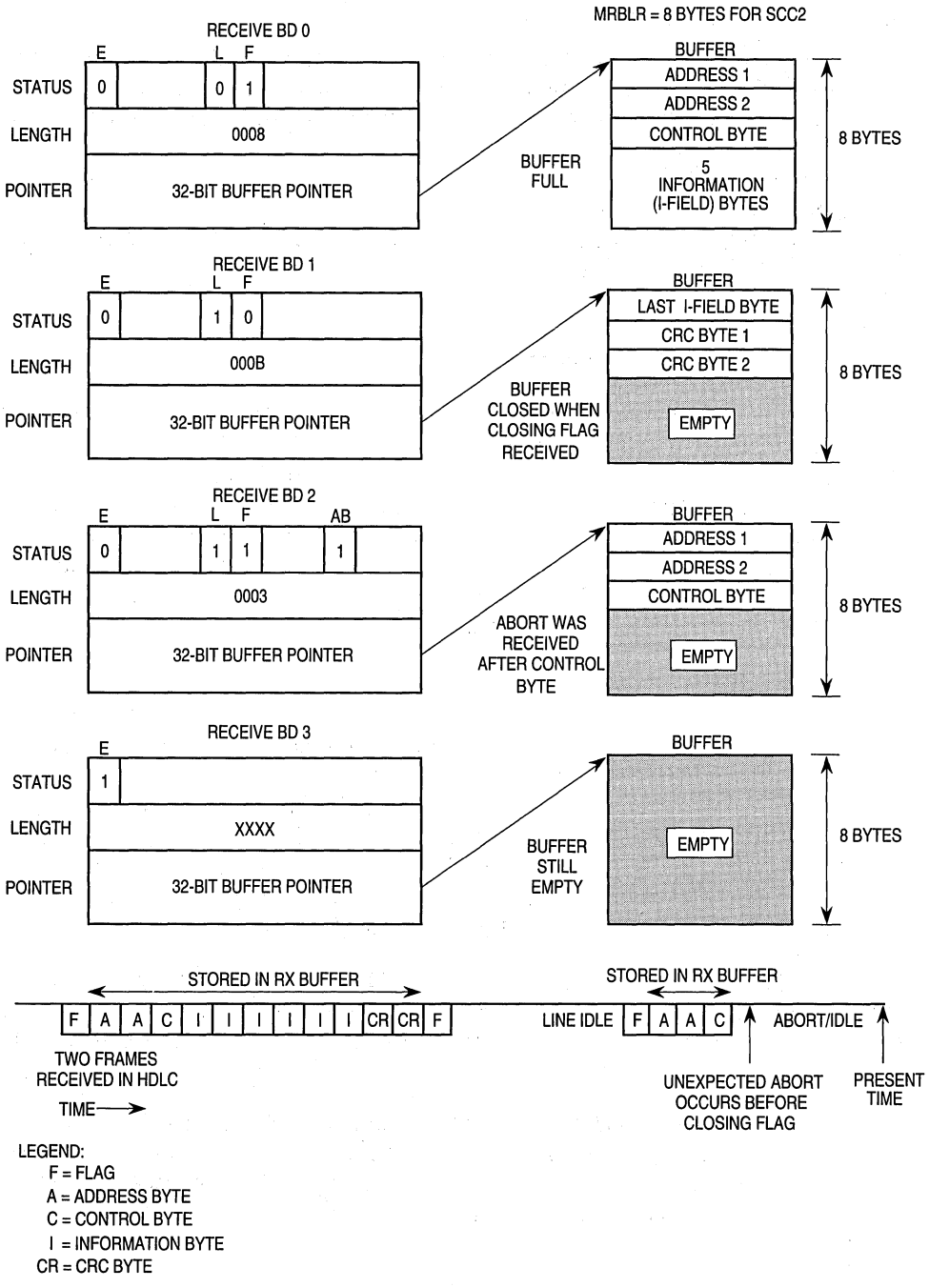


Figure 16-78. HDLC Receive Buffer Descriptor Example

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	**E**	RES	**W**	**I**	L	F	**CM**	RES	DE	RES	LG	NO	AB	CR	OV	CD
OFFSET + 2	**DATA LENGTH**															
OFFSET + 4	**RX DATA BUFFER POINTER**															
OFFSET + 6																

NOTE: You are only responsible for initializing the items in bold.

E—Empty

0 = The data buffer associated with this RX buffer descriptor has been filled with data or reception has been aborted because of an error condition. The core is free to examine or write to any fields of this RX buffer descriptor. The communication processor module does not use this buffer descriptor as long as the E bit is zero.

1 = The data buffer associated with this buffer descriptor is empty or is currently receiving data. This RX buffer descriptor and its associated receive buffer are owned by the communication processor module. Once the E bit is set, the core should not write any fields of this RX buffer descriptor.

Bits 1, 7, and 9—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the RX buffer descriptor table.

1 = This is the last buffer descriptor in the RX buffer descriptor table. After this buffer is used, the communication processor module receives incoming data into the first buffer descriptor that RBASE points to in the table. The number of RX buffer descriptors in this table are programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0 = The RXB bit is not set after this buffer is used, but RXF operation remains unaffected.

1 = The RXB or RXF bit in the SCCE–HDLC register is set when the SCC2 HDLC controller uses this buffer. These two bits can cause interrupts if they are enabled.

L—Last in Frame

This bit is set by the SCC2 HDLC controller when this buffer is the last one in a frame. This implies the reception of a closing flag or reception of an error, in which case one or more of the CD, OV, AB, and LG bits are set. The SCC2 HDLC controller writes the number of frame octets to the data length field.

0 = This buffer is not the last one in a frame.

1 = This buffer is the last one in a frame.

F—First in Frame

This bit is set by the SCC2 HDLC controller when this buffer is the first in a frame.

0 = The buffer is not the first one in a frame.
1 = The buffer is the first one in a frame.

CM—Continuous Mode

0 = Normal operation.
1 = The E bit is not cleared by the communication processor module after this buffer descriptor is closed, thus allowing the associated data buffer to be automatically overwritten next time the communication processor module accesses this buffer descriptor. The E bit is cleared if an error occurs during reception, regardless of how the CM bit is set.

DE—DPLL Error

This bit is set by the SCC2 HDLC controller when a DPLL error occurs while this buffer is being received. In decoding modes where a transition is promised every bit, the DE bit is set when a missing transition occurs.

LG—RX Frame Length Violation

This bit indicates when a frame length greater than the maximum defined for this channel is recognized and only the maximum-allowed number of bytes (MFLR) is written to the data buffer. This event is not reported until the RX buffer descriptor is closed, the RXF bit is set, and the closing flag is received. The actual number of bytes received between flags is written to the data length field of this buffer descriptor.

NO—RX Nonoctet Aligned Frame

This bit indicates when a frame is received that contains a number of bits almost divisible by eight.

AB—RX Abort Sequence

This bit indicates when a minimum of seven consecutive ones are received when a frame is received.

CR—RX CRC Error

This bit indicates that this frame contains a CRC error. The received CRC bytes are always written to the receive buffer.

OV—Overrun

This bit indicates that a receiver overrun has occurred while a frame was being received.

CD—Carrier Detect Lost

This bit indicates that a carrier detect signal has been negated while a frame was being received. This bit is only valid when working in NMSI mode.

Data Length

These bits represent the number of octets the communication processor module writes into this buffer descriptor data buffer. It is written by the communication processor module once the buffer descriptor is closed. When this buffer descriptor is the last buffer descriptor in the frame (L = 1), the data length contains the total number of frame octets, including 2 or 4 bytes for CRC. The actual amount of memory allocated for this buffer should be greater than or equal to the MRBLR.

RX Data Buffer Pointer

These bits always point to the first location of the associated data buffer, they reside in internal or external memory, and must be divisible by four.

16.9.16.10 SCC2 HDLC TRANSMIT BUFFER DESCRIPTOR. Data is sent to the SCC2 HDLC controller for transmission on an SCC2 channel by arranging it in buffers referenced by the channel's TX buffer descriptor table. Using the transmit (TX) buffer descriptors, the communication processor module confirms transmission and indicates error conditions so that the core knows the buffers have been serviced.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	**R**	RES	**W**	**I**	**L**	**TC**	**CM**	RESERVED							UN	CT
OFFSET + 2	**DATA LENGTH**															
OFFSET + 4	**TX DATA BUFFER POINTER**															
OFFSET + 6																

NOTE: You are only responsible for initializing the items in bold.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

R—Ready

0 = The data buffer associated with this buffer descriptor is not ready for transmission. You are free to manipulate this buffer descriptor or its associated data buffer. The communication processor module clears this bit after the buffer has been transmitted or an error condition is encountered.

1 = The data buffer, which you have prepared, has not been transmitted or is currently being transmitted. You cannot write any fields of this buffer descriptor once this bit is set.

Bits 1 and 7–13—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the TX buffer descriptor table.
1 = This is the last buffer descriptor in the TX buffer descriptor table. After this buffer has been used, the communication processor module transmits data from the first buffer descriptor that TBASE points to in the table. The number of TX buffer descriptors in this table are programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is serviced.
1 = The TXB or TXE bit in the SCCE–HDLC register is set when this buffer is serviced by the SCC2 HDLC controller. These bits can cause interrupts if they are enabled.

L—Last

0 = This is not the last buffer in the frame.
1 = This is the last buffer in the frame.

TC—TX CRC

This bit is valid only when the L bit is set. Otherwise, it is ignored.

0 = Transmit the closing flag after the last data byte. This setting can be used to send a bad CRC after the data for testing purposes.
1 = Transmit the CRC sequence after the last data byte.

CM—Continuous Mode

0 = Normal operation.
1 = The R bit is not cleared by the communication processor module after this buffer descriptor is closed, thus allowing the associated data buffer to be automatically retransmitted next time the communication processor module accesses this buffer descriptor. However, the R bit is cleared if an error occurs during transmission, regardless of how the CM bit is set.

UN—Underrun

This bit indicates when an SCC2 HDLC controller encounters a transmitter underrun condition while transmitting the associated data buffer. The SCC2 HDLC controller writes these bits after it finishes transmitting the associated data buffer.

CT—$\overline{\text{CTS}}$ Lost

This bit indicates when $\overline{\text{CTS}}$ in NMSI mode or layer 1 grant is lost in GCI mode during frame transmission. If data from more than one buffer is currently in the FIFO when this error occurs, this bit is set in the currently open TX buffer descriptor. The SCC2 HDLC controller writes these bits after it finishes transmitting the associated data buffer.

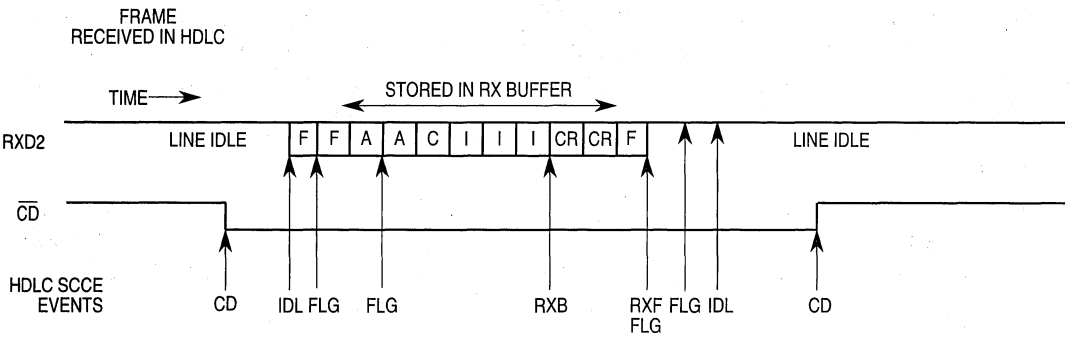
DATA LENGTH

These bits represent the number of bytes the SCC2 HDLC controller transmits from this buffer descriptor data buffer. It is never modified by the communication processor module. The value of this field should be greater than zero. The SCC2 HDLC controller writes these bits after it finishes transmitting the associated data buffer.

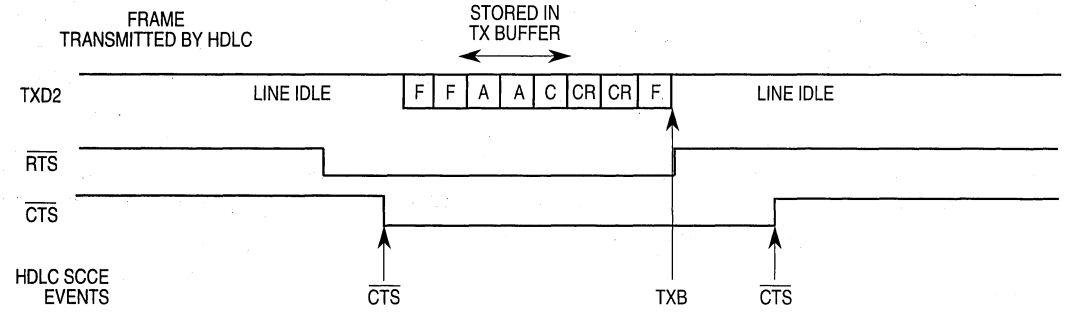
TX DATA BUFFER POINTER

These bits contain the address of the associated data buffer and can be even or odd. The buffer can reside in internal or external memory. This value is never modified by the communication processor module. The SCC2 HDLC controller writes these bits after it finishes transmitting the associated data buffer.

16.9.16.11 SCC2 HDLC EVENT REGISTER. When the SCC2 is in HDLC mode, the 16-bit, memory-mapped SCC2 event register is referred to as the SCC2 HDLC event register (SCCE–HDLC). Since each protocol has specific requirements, the SCCE bits are different for each implementation. This register is used to report events recognized by the HDLC channel and to generate interrupts. When an event is recognized, the SCC2 HDLC controller sets the corresponding bit in this register. Interrupts generated by this register can be masked in the SCCM–HDLC register. An example of interrupts that can be generated using the HDLC protocol is illustrated in Figure 16-79.



NOTES:
1. RXB event assumes receive buffers are 6 bytes each.
2. The second IDL event occurs after 15 ones are received in a row.
3. The FLG interrupts show the beginning and end of flag reception.
4. The FLG interrupt at the end of the frame may precede the RXF interrupt due to receive FIFO latency.
5. The CD event must be programmed in the port C parallel I/O, not in SCC2 itself.
6. F = flag, A = address byte, C = control byte, I = information byte, and CR = CRC byte.



NOTES:
1. TXB event shown assumes all three bytes were put into a single buffer.
2. Example shows one additional opening flag. This is programmable.
3. The CTS event must be programmed in the port C parallel I/O, not in SCC2 itself.

Figure 16-79. HDLC Interrupt Event Example

A bit is cleared by writing a 1 (writing a zero has no effect) and more than one bit can be cleared at a time. Also, all unmasked bits must be cleared before the communication processor module clears the internal interrupt request. This register is cleared at reset and can be read at any time.

SCCE–HDLC

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED			GLR	GLT	DCC	FLG	IDL	GRA	RESERVED		TXE	RXF	BSY	TXB	RXB
RESET	0			0	0	0	0	0	0	0		0	0	0	0	0
R/W	R/W			R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA30															

Bits 0–2 and 9–10—Reserved

These bits are reserved and should be set to 0.

GLR—Glitch on RX

This bit indicates that the serial communication controller has detected a clock glitch on the receive clock.

GLT—Glitch on TX

This bit indicates that the serial communication controller has detected a clock glitch on the transmit clock.

DCC—DPLL Carrier-Sense Changed

This bit indicates that the carrier-sense status generated by the DPLL has changed state. Its real-time status can be found in the SCCS–HDLC register. This is not the $\overline{CD}$ pin status that is reported in port C and it is only valid when the DPLL is used.

FLG—Flag Status

This bit indicates when the SCC2 HDLC controller stops or starts receiving HDLC flags. Its real-time status can be obtained in the SCCS–HDLC register.

IDL—Idle Sequence Status Changed

This bit indicates when a change in the status of the serial line occurs on the HDLC line. The real-time status of the line can be read in the SCCS–HDLC register.

GRA—Graceful Stop Complete

this bit indicates that a graceful stop, which was initiated by the **GRACEFUL STOP TRANSMIT** command, has completed. This bit is set as soon as the transmitter finishes transmitting any frame that was in progress when the command was issued. It is set immediately if no frame was in progress when the command was issued.

TXE—TX Error

This bit indicates that an error has occurred on the transmitter channel.

RXF—RX Frame

This bit indicates when a complete frame is received on the HDLC channel. This bit is set no sooner than two clocks after the last bit of the closing flag is received.

BSY—Busy Condition

This bit indicates when a frame has been received and discarded due to a lack of buffers.

TXB—Transmit Buffer

This bit indicates that a buffer has been transmitted on the HDLC channel. This bit is set no sooner than when the last bit of the closing flag begins its transmission if the buffer is the last one in the frame. Otherwise, this bit is set after the last byte of the buffer is written to the transmit FIFO.

RXB—Receive Buffer

This bit indicates that the HDLC channel received a buffer that is not a complete frame.

16.9.16.12 SCC2 HDLC MASK REGISTER. When the SCC2 is in HDLC mode, the 16-bit, read/write SCC2 mask register is referred to as the SCC2 HDLC mask register (SCCM–HDLC). It has the same bit formats as the SCCE–HDLC register. If a bit in this register is a 1, the corresponding interrupt in the SCCE–HDLC register is enabled. If the bit is zero, the corresponding interrupt in the SCCE–HDLC is masked.

SCCM–HDLC

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED			GLR	GLT	DCC	FLG	IDL	GRA	RESERVED		TXE	RXF	BSY	TXB	RXB
RESET	0			0	0	0	0	0	0	0		0	0	0	0	0
R/W	R/W			R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA34															

16.9.16.13 SCC2 HDLC STATUS REGISTER. The 8-bit read-only SCC2 status register is referred to as the SCC2 HDLC status register (SCCS–HDLC). Since each protocol has specific requirements, the SCCS bits are different for each implementation. This register allows you to monitor real-time status conditions on the RXD2 line. The real-time status of the $\overline{CTS}$ and $\overline{CD}$ pins are part of the port C parallel I/O.

SCCS–HDLC

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED					FG	CS	ID
RESET	0					0	0	0
R/W	R					RO	R	R
ADDR	(IMMR & 0xFFFF0000) + 0xA37							

Bits 0–4—Reserved

These bits are reserved and should be set to 0.

FG—Flags

When this bit is cleared, the most recently received 8 bits are examined every bit time to see if a flag is present. FG is set as soon as an HDLC flag (0x7E) is received on the line and once it is set, it remains set at least 8 bit times and the next eight received bits are examined. If another flag occurs, this bit stays set for at least another eight bits. Otherwise, it is cleared and the search begins again. The line is checked after the data has been decoded by the DPLL.

0 = HDLC flags are not being received.
1 = HDLC flags are being received.

CS—Carrier Sense (DPLL)

This bit shows the real-time carrier sense of the line as determined by the DPLL.

0 = The DPLL does not sense a carrier.
1 = The DPLL senses a carrier.

ID—Idle Status

This bit is set when the RXD2 pin is a logic one for 15 or more consecutive bit times. It is cleared after a single logic zero is received.

0 = The pin is busy.
1 = The pin is idle.

16.9.16.14 SCC2 HDLC PROGRAMMING EXAMPLE #1. The following initialization sequence is for an SCC2 HDLC channel with an external clock. The SCC2 in HDLC mode is configured with the $\overline{RTS2}$, $\overline{CTS2}$, and $\overline{CD2}$ pins active and the CLK3 pin is used for both the HDLC receiver and transmitter.

1. Configure the port A pins to enable the TXD2 and RXD2 pins. Write PAPAR bits 13 and 12 with ones and then write PADIR and PAODR bits 13 and 12 with zeros.
2. Configure the port C pins to enable $\overline{RTS2}$, $\overline{CTS2}$, and $\overline{CD2}$. Write PCPAR bit 14 with one, and bits 9 and 8 with zeros, PCDIR bits 14, 9, and 8 with zeros, and PCSO bits 9 and 8 with ones.
3. Configure port A to enable the CLK3 pin. Write PAPAR bit 5 with a one and PADIR bit 5 with a zero.
4. Connect the CLK3 pin to SCC2 using the serial interface. Write the R2CS and T2CS bits in the SICR to 110.
5. Connect the SCC2 to the NMSI (its own set of pins) and clear the SC2 bit in the SICR.
6. Write 0x0001 to the SDCR to set the SDMA bus arbitration level to 5.
7. Write RBASE and TBASE in the SCC2 parameter RAM to point to the RX buffer descriptor and TX buffer descriptors in the dual-port RAM. Assuming one RX buffer descriptor at the beginning of dual-port RAM and one TX buffer descriptor following it, write RBASE with 0x2000 and TBASE with 0x2008.
8. Program the CPCR to execute the **INIT RX AND TX PARAMS** command for the serial communication controller. This command causes the RBPTR and TBPTR parameters of the serial channel to be updated with the new values just programmed into RBASE and TBASE.
9. Write 0x18 to RFCR and 0x18 to TFCR for normal operation.
10. Write MRBLR with the maximum number of bytes per receive buffer. Assume 256 bytes, so MRBLR = 0x0100. The value 256 is chosen so that an entire receive frame can fit into one receive buffer.
11. Write 0x0000F0B8 to C_MASK to comply with 16-bit CCITT-CRC.
12. Write 0x0000FFFF to C_PRES to comply with 16-bit CCITT-CRC.
13. Clear DISFC, CRCEC, ABTSC, NMARC, and RETRC for clarity.
14. Write 0x0100 to MFLR so the maximum frame size is 256 bytes.
15. Write 0x0001 to RFTHR to allow interrupts after each frame.
16. Write 0x0000 to HMASK to allow all addresses to be recognized.
17. Clear HADDR1, HADDR2, HADDR3, and HADDR4 for clarity.
18. Initialize the RX buffer descriptor. Assume the RX data buffer is at 0x00001000 in main memory. Write 0xB000 to RX_BD_Status, write 0x0000 to RX_BD_Length (not required), and 0x00001000 to RX_BD_Pointer.
19. Initialize the TX buffer descriptor. Assume the TX data frame is at 0x00002000 in main memory and contains five 8-bit characters. Write 0xBC00 to TX_BD_Status, 0x0005 to TX_BD_Length, and 0x00002000 to TX_BD_Pointer.

20. Write 0xFFFF to the SCCE–HDLC to clear any previous events.
21. Write 0x001A to the SCCM–HDLC to enable the TXE, RXF, and TXB interrupts.
22. Write 0x20000000 to the CIMR to allow SCC2 to generate a system interrupt. The CICR should also be initialized.
23. Write 0x00000000 to the GSMR_H to enable normal behavior of the $\overline{CTS}$ and $\overline{CD}$ pins and idles between frames (as opposed to flags).
24. Write 0x00000000 to the GSMR_L to configure the $\overline{CTS}$ and $\overline{CD}$ pins to automatically control transmission and reception and the HDLC mode. Normal operation of the transmit clock is used. Notice that the transmitter (ENT) and receiver (ENR) have not been enabled. If inverted HDLC operation is preferred, set the RINV and TINV bits in the GSMR_L.
25. Set the PSMR–HDLC to 0x0000 to configure one opening and one closing flag, 16-bit CCITT-CRC, and prevention of multiple frames in the FIFO.
26. Write 0x00000030 to the GSMR_L to enable the SCC2 transmitter and receiver. This additional write ensures that the ENT and ENR bits are enabled last.

Note: After 5 bytes and CRC have been transmitted, the TX buffer descriptor is automatically closed. Once a complete frame is received, the RX buffer descriptor is closed. Any data received after 256 bytes or a single frame causes a busy (out-of-buffers) condition since only one RX buffer descriptor is prepared.

16.9.16.15 SCC2 HDLC PROGRAMMING EXAMPLE #2. The following initialization sequence is for an SCC2 HDLC channel that uses the DPLL in a Manchester encoding. You must provide a clock that is 16× the preferred bit rate on the CLK3 pin. CLK3 is then connected to the HDLC transmitter and receiver. A baud rate generator could also be used. The SCC2 HDLC controller is configured with the $\overline{RTS2}$, $\overline{CTS2}$, and $\overline{CD2}$ pins active.

1. Follow steps 1 through 22 in example #1 above.
2. Write 0x00000000 to the GSMR_H to enable normal behavior of the $\overline{CTS}$ and $\overline{CD}$ pins and idles between frames.
3. Write 0x004AA400 to the GSMR_L to configure the carrier sense as always active, a 16-bit preamble of "01" pattern, 16× operation of the DPLL for the receiver and transmitter, Manchester encoding for the receiver and transmitter, and HDLC mode. The $\overline{CTS}$ and $\overline{CD}$ pins should also be configured to automatically control transmission and reception. Notice that the transmitter (ENT) and receiver (ENR) have not been enabled yet.
4. Set the PSMR–HDLC to 0x0000 to configure one opening and one closing flag, 16-bit CCITT-CRC. Multiple frames in the FIFO are not allowed in this example.
5. Write 0x004AA430 to the GSMR_L to enable the SCC2 transmitter and receiver. This additional write ensures that the ENT and ENR bits are enabled last.

Note: After 5 bytes in the preamble have been transmitted, the TX buffer descriptor is automatically closed. Once 16 bytes have been received, the RX buffer descriptor is closed. Any data received after 16 bytes causes a busy (out-of-buffers) condition since only one RX buffer descriptor is prepared.

16.9.17 The HDLC Bus Controller

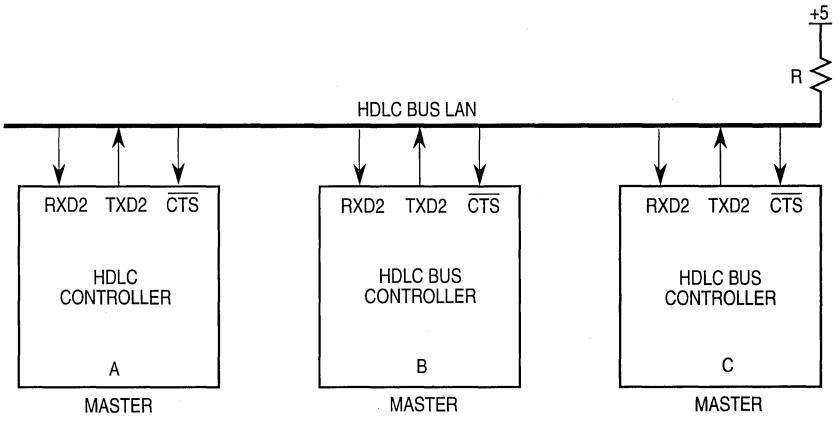
The HDLC bus is an enhancement that makes it easy to implement an HDLC-based LAN and other point-to-multipoint configurations. Most HDLC-based controllers only provide point-to-point communications. The HDLC bus is based on the techniques used in the CCITT ISDN I.430 and ANSI T1.605 standards for D channel point-to-multipoint operation over the S/T interface. However, the HDLC bus is not fully compliant with I.430 or T1.605 and cannot be used to replace devices that implement these protocols. Instead, it is more suited to fulfill the needs of non-ISDN LAN and point-to-multipoint configurations.

You should review the basic features of the I.430 and T1.605 before learning about the HDLC bus. The I.430 and T1.605 define a method whereby eight terminals can be connected over the D-channel of the S/T bus of ISDN. The protocol used at layer 2 is a variant of HDLC, called LAPD. However, at layer 1, a method is provided that allows the eight terminals to access the physical S/T bus so they can send frames to the switch.

To find out if a channel is clear, the S/T interface device looks at an "echo" bit on the line designed to echo the last bit transmitted on the D channel. Depending on the "class" of the terminal and the particular situation, the S/T interface device can wait for 7, 8, 9, or 10 ones on the echo bit before allowing the LAPD frame to begin transmission. Once transmission begins, the S/T chip monitors the data that is sent and if the echo bit matches the transmitted data, transmission continues. If the echo bit is ever zero when the transmit bit is 1, then a collision will occur between terminals and the station that transmitted a zero will no longer transmit. The station that transmitted a one, however, continues as normal.

The I.430 and T1.605 provide a physical layer protocol that allows multiple terminals to share the same physical connection. Where collisions are concerned, these protocols use the bus efficiently because one station is always able to complete its transmission. Once a station completes a transmission, it lowers its own priority to give other devices fair access to the physical connection.

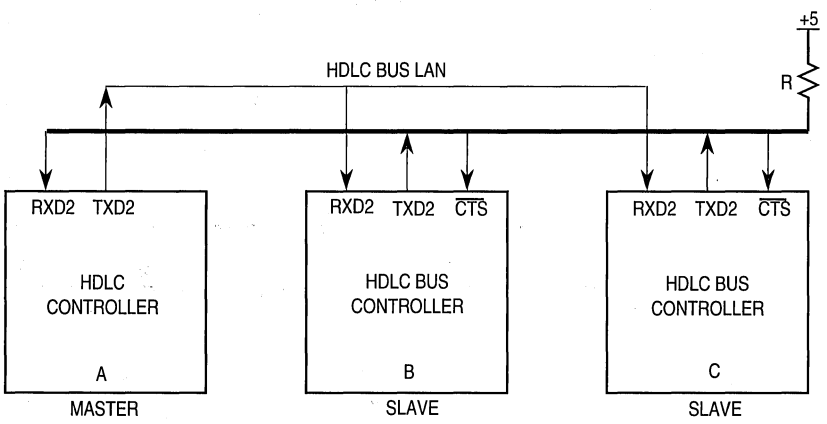
The HDLC bus works pretty much the same way, except for a few differences. First, the HDLC bus does not use the echo bit, but rather a separate pin to monitor the data that is transmitted. The transmitted data is simply connected to the $\overline{CTS}$ input. Second, the HDLC bus is a synchronous digital open-drain connection for short-distance configurations, rather than the more complex configurations of an S/T interface. Third, the HDLC bus allows any HDLC-based frame protocol to be implemented at layer 2, not just LAPD. Fourth, the HDLC bus devices wait either 8 or 10 bit times before transmitting, rather than 7, 8, 9, or 10 bits. Figure 16-80 illustrates HDLC bus in its most common LAN configuration, the multimaster configuration. All stations can transmit and receive data to or from every other station on the LAN and all transmissions are half-duplex, which is typical in LANs.



NOTES:
1. Transceivers may be used to extend the LAN size, if necessary.
2. The TXD2 pins of slave devices should be configured to open-drain in the port C parallel I/O port.

Figure 16-80. Typical HDLC Bus Multimaster Configuration

In single-master configuration, a master station transmits to any slave station without any collisions. The slaves only communicate with the master, but can experience collisions in their access over the bus. In this configuration, a slave that communicates with another slave must first transmit its data to the master, where the data is buffered in RAM and then retransmitted to the other slave. The benefit of this configuration, however, is that full-duplex operation can be obtained. In a point-to-multipoint environment, this is the preferred configuration. Figure 16-81 illustrates the single-master configuration.



NOTES:
1. Transceivers may be used to extend the LAN size, if necessary.
2. The TXD2 pins of slave devices should be configured to open-drain in the port C parallel I/O port.

Figure 16-81. Typical HDLC Bus Single-Master Configuration

16.9.17.1 FEATURES. The following list contains the main features of the HDLC bus:

- Superset of the SCC2 in HDLC mode features
- Automatic HDLC bus access
- Automatic retransmission in case of a collision
- May be used with the NMSI mode or a TDM bus
- Delayed $\overline{\text{RTS}}$ mode

16.9.17.2 ACCESSING THE HDLC BUS. The HDLC bus ensures an orderly access to the bus when two or more transmitters attempt to access it simultaneously. One transmitter is always successful in completing its transmission. While in the active condition, the HDLC bus controller monitors the bus using the $\overline{CTS}$ pin. It counts the number of one bits using the $\overline{CTS}$ pin and if a zero is detected, the internal counter is cleared. Once eight consecutive ones have been received, the HDLC bus controller starts transmitting on the line. While it is transmitting information on the bus, the data is continuously compared with the $\overline{CTS}$ pin and used to sample the external bus. The $\overline{CTS}$ sample is taken halfway through the bit time using the rising edge of the transmit clock. If the transmitted bit is the same as the received $\overline{CTS}$ sample, the HDLC bus controller continues its transmission. If, however, the received $\overline{CTS}$ sample is zero, but the transmitted bit was 1, the HDLC bus controller stops transmitting after that bit and returns to active condition. Since the HDLC bus uses a wired-OR scheme, a transmitted zero has priority over a transmitted 1. Figure 16-82 illustrates how the $\overline{CTS}$ pin is used.

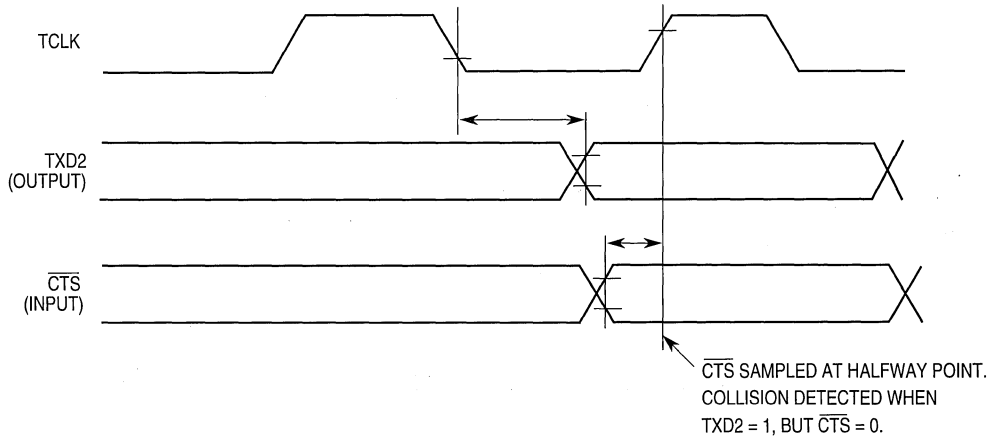


Figure 16-82. Detecting an HDLC Bus Collision

If the source address is included in the HDLC frame and destination address, a predefined priority of nodes results. Collisions can be detected no later than the end of the source address, if one is included.

Note: The HDLC bus can be used with many different HDLC-based frame formats.

To ensure that all stations gain an equal share of the bus, a priority mechanism is implemented on the HDLC bus. Once an HDLC bus node has finished transmitting a frame, it waits 10 consecutive one bits, instead of eight, before beginning the next transmission. Using this method, all nodes that need to transmit can obtain the bus before a node transmits twice. Once a node finds out that 10 consecutive ones have occurred on the bus, it tries to transmit and reinstate its original priority of waiting for eight ones.

16.9.17.2.1 Improving Performance. Since the HDLC bus is used in a wired-OR configuration, the limitations of the HDLC bus is determined by the rise time of the one bit. Figure 16-83 illustrates a method to increase performance. You must supply a clock that is lower more than it is high, which allows the one bit to have more rise time.

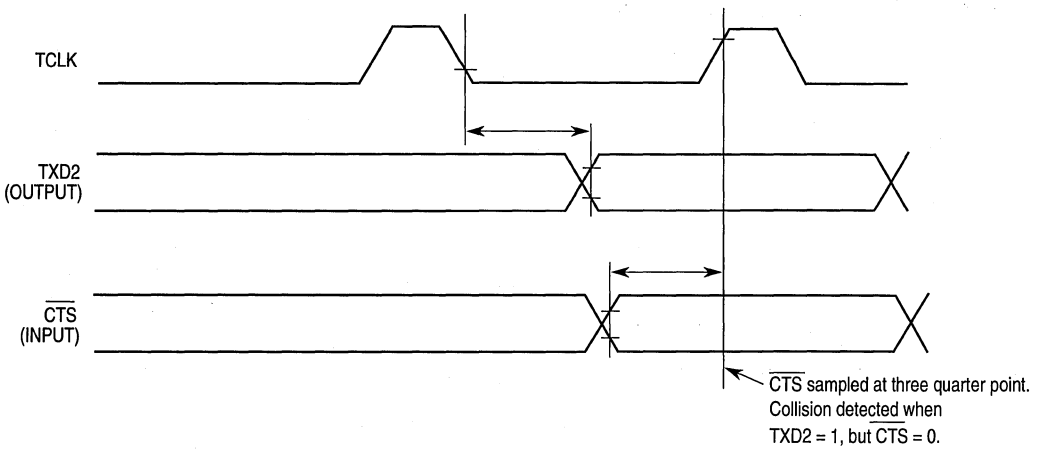
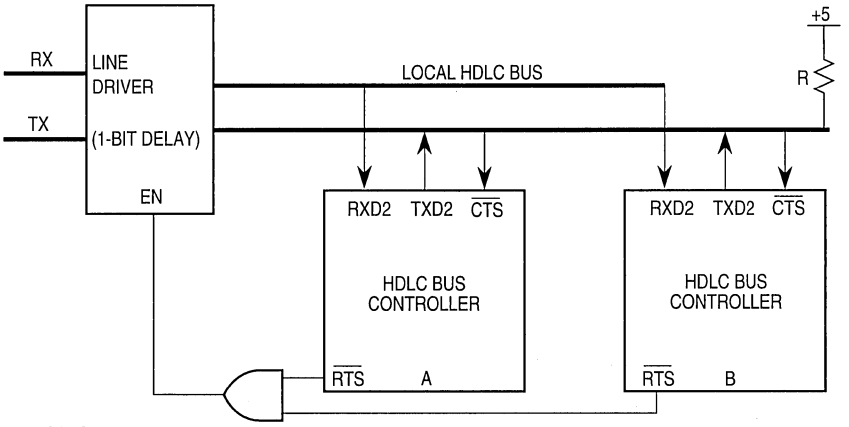


Figure 16-83. Example of a Nonsymmetrical Duty Cycle

16.9.17.2.2 Delaying $\overline{\text{RTS}}$ Mode. Sometimes the HDLC bus can be used in a configuration with a local HDLC bus and a standard transmission line that is not an HDLC bus. This situation is illustrated in Figure 16-84. The local HDLC bus controllers do not communicate with each other, but with a station on the transmission line; yet the HDLC bus protocol is used to control access to the transmission line.



NOTES:
1. The TXD2 pins of slave devices should be configured to open-drain in the port C parallel I/O port.
2. The $\overline{\text{RTS}}$ pins of each HDLC bus controller are configured to delayed $\overline{\text{RTS}}$ mode.

Figure 16-84. HDLC Bus Transmission Line Configuration

Normally, the $\overline{\text{RTS}}$ pin goes active at the beginning of the opening flag's first bit. Although using the $\overline{\text{RTS}}$ pin is not required, there is a mode on the MPC823 HDLC bus that delays the $\overline{\text{RTS}}$ signal by one bit. This mode is selected with the BRM bit in the PSMR–SCC2 HDLC register.

Delayed $\overline{\text{RTS}}$ mode is useful when the HDLC bus connects multiple local nodes to a transmission line. If the transmission line driver has a one-bit delay, then the delayed $\overline{\text{RTS}}$ line can be used to enable the output of the transmission line driver. The result is that the transmission line bits always drive "clean" and without any collisions. $\overline{\text{RTS}}$ timing is illustrated in Figure 16-85.

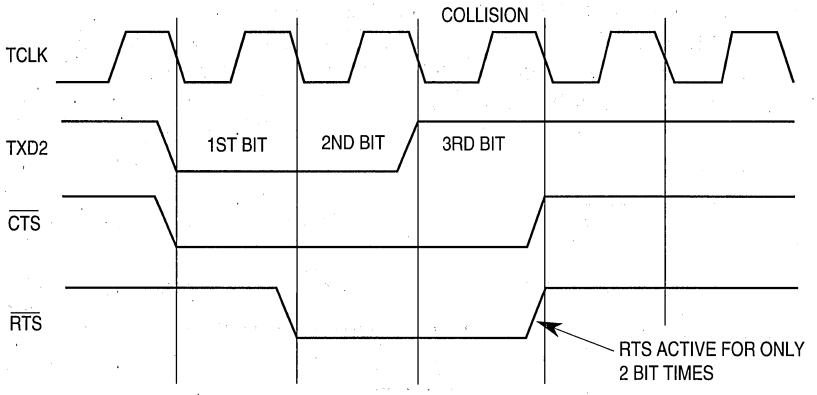
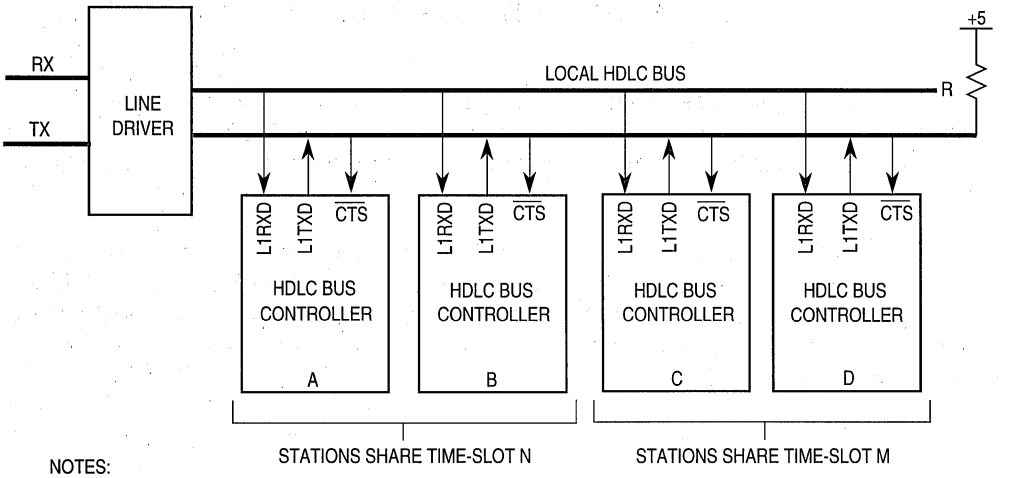


Figure 16-85. Delayed RTS Mode

16.9.17.2.3 Using the Time-Slot Assigner. Sometimes the HDLC bus can be used in a configuration that has a local HDLC bus and a time-division multiplex transmission line that is not an HDLC bus. Figure 16-86 illustrates such a case. The local HDLC bus controllers all communicate over time-slots. However, more than one HDLC bus controller is assigned to a given time-slot and the HDLC bus protocol controls access during that time-slot.



NOTES:
1. All TX pins of slave devices should be configured to open-drain in the port C parallel I/O port.
2. The TSA in the serial interface of each station is used to configure the preferred time-slot.
3. You can choose the number of stations to share a time-slot. In this example, two are used.

Figure 16-86. HDLC Bus Time-Slot Assigner Transmission Line Configuration

If the serial communication controller is configured to operate using the time-slot assigner of the serial interface, then the data is received and transmitted using the L1TXDA and L1RXDA pins. The collision sensing is still obtained from the individual serial communication controller $\overline{\text{CTS}}$ pin, so the $\overline{\text{CTS}}$ pin must be configured in port C to connect to the preferred serial communication controller. Since the serial communication controller only receives clocks during its time-slot, the $\overline{\text{CTS}}$ pin is only sampled during the transmit clock edges of the particular serial communication controller time-slot.

16.9.17.3 HDLC BUS MEMORY MAP AND PROGRAMMING. The HDLC bus on the MPC823 is implemented using the SCC2 in HDLC mode with certain bits set. If you want to do otherwise, see **Section 16.9.16.5 Programming the SCC2 in HDLC Mode** for information about HDLC mode programming.

Use the general SCC2 mode high and low registers (GSMR_x) described in **Section 16.9.2 The General SCC2 Mode Registers** to program the HDLC bus controller with the following steps:

1. Set the MODE field to HDLC mode in the GSMR_L.
2. Set the CTSS field to 1 and all other bits to zero or default in the GSMR_H.
3. Set the DIAG field for normal operation in the GSMR_L.
4. Set the RDCR and TDCR bits for 1× clock in the GSMR_L.
5. Set the TENC and RENC bits for NRZ in the GSMR_L.
6. Clear the RTSM bit in the GSMR_H.
7. Set the ENT and ENR bits in the GSMR_L, as preferred.

Use the SCC2 protocol-specific mode register described in **Section 16.9.16.8 SCC2 HDLC Mode Register** to program the HDLC bus controller with the following steps:

1. Set the NOF field as preferred.
2. Set the CRC field to 16-bit CRC CCITT.
3. Set the RTE bit.
4. Set the BUS bit.
5. Set the BRM bit to 1 or zero as preferred.
6. Set all other bits to zero or default.

16.9.17.3.1 HDLC Bus Controller Programming Example. Except for the previously discussed GSMR_x and PSMR–SCC2 HDLC programming, use the example in **Section 16.9.16.14 SCC2 HDLC Programming Example #1**.

16.9.18 The SCC2 in AppleTalk Mode

AppleTalk® is a set of protocols developed by Apple Computer Inc. to provide a LAN service between Macintosh computers and printers. Although AppleTalk can be implemented over a variety of physical and link layers, including Ethernet, the AppleTalk protocols have traditionally been more closely associated with one particular physical and link layer protocol called LocalTalk.

LocalTalk refers to an HDLC-based link and physical layer protocol that runs at the rate of 230.4kbps. In this manual, the term AppleTalk controller refers to a support that the MPC823 provides for the LocalTalk protocol. The AppleTalk controller provides the required frame synchronization, bit sequence, preamble, and postamble onto standard HDLC frames. These capabilities, as well as the use of the HDLC controller in conjunction with DPLL operation in FM0 mode, provide the proper connection formats to the LocalTalk bus.

16.9.18.1 OPERATING THE LOCALTALK BUS. A LocalTalk frame is basically a modified HDLC frame.

SYNC SEQ	HDLC FLAGS	DEST ADDR	SOURCE ADDR	CONTROL BYTE	DATA (OPTIONAL)	CRC-16	CLOSING FLAG	ABORT SEQUENCE
> 3 BITS	2 OR MORE BYTES	1 BYTE	1 BYTE	1 BYTE	0–600 BYTES	2 BYTES	1 BYTE	12–18 ONES

Figure 16-87. LocalTalk Frame Format

The frame begins when a synchronization sequence greater than 3 bits is sent. This sequence consists of at least one logical 1 bit (FM0-encoded) followed by greater than 2 bit times of line idle with no particular maximum time specified. The idle time allows LocalTalk equipment to sense a carrier by detecting a "missing clock" on the line. The remainder of the frame is a typical half-duplex HDLC frame. Two or more flags are sent, which allows for bit, byte, and frame delineation or detection. Then, two bytes of address, destination, and source are transmitted, followed by a byte of control and 0 to 600 data bytes. Next, two bytes of CRC (described in **Section 16.9.16.8 SCC2 HDLC Mode Register**) are sent. The LocalTalk frame is then terminated by a flag and a restricted HDLC abort sequence. Then the transmitter's driver is disabled.

The control byte within the LocalTalk frame indicates the type of frame. Control byte values that range from 0x01 to 0x7f are data frames and control byte values that range from 0x80 to 0xff are control frames. There are four types of control frames:

- ENQ—Enquiry
- ACK—Enquiry acknowledgment
- RTS—Request to send a data frame
- CTS—Clear to send a data frame

Frames are sent in groups called dialogs, which are handled by the software. For instance, to transfer a data frame, three frames are actually sent over the network. An RTS frame (not to be confused with the RS-232 $\overline{RTS}$ pin) is sent to request the network, then a CTS frame is sent by the destination node, and the data frame is sent by the requesting node. These three frames comprise one possible type of dialog. Once a dialog has begun, other nodes cannot start transmitting until the dialog is complete. Frames within a dialog are transmitted with a maximum interframe gap (IFG) of 200 microseconds. Although the LocalTalk specification does not state it, there is also a minimum recommended IFG of 50ms. Dialogs must be separated by a minimum interdialog gap (IDG) of 400ms. These gaps are implemented by the software.

Depending on the protocol, collisions should only be encountered during RTS and ENQ frames. Once frame transmission begins, it is fully transmitted, regardless of whether it collides with another frame. ENQ frames are infrequent and only sent when a node powers up and enters the network. A higher level protocol controls the uniqueness and transmission of ENQ frames.

In addition to the frame fields, LocalTalk requires that the frame be FM0 (differential Manchester space) encoded, which requires one level transition on every bit boundary. If the value to be encoded is a logic zero, FM0 requires a second transition in the middle of the bit time. The purpose of FM0 encoding is to avoid having to transmit clocking information on a separate wire. With FM0, the clocking information is present whenever valid data is present.

16.9.18.2 FEATURES. The following list summarizes the features of the SCC2 in AppleTalk mode:

- Superset of the HDLC controller features
- FM0 encoding/decoding
- Programmable transmission of sync sequence
- Automatic postamble transmission
- Reception of sync sequence does not cause extra $\overline{CD}$ interrupts
- Reception is automatically disabled while transmitting a frame
- Transmit-on-demand feature that expedites frames
- Connects directly to an RS-422 transceiver

16.9.18.3 CONNECTING TO APPLETALK. The MPC823 connects to LocalTalk and, using the TXD2, $\overline{\text{RTS}}$, and RXD2 pins, is an interface for the RS-422 transceiver. The RS-422, in turn, is an interface for the LocalTalk connector. Although it is not shown, a passive RC circuit is recommended between the transceiver and connector. Figure 16-88 illustrates how to connect the MPC823 to LocalTalk.

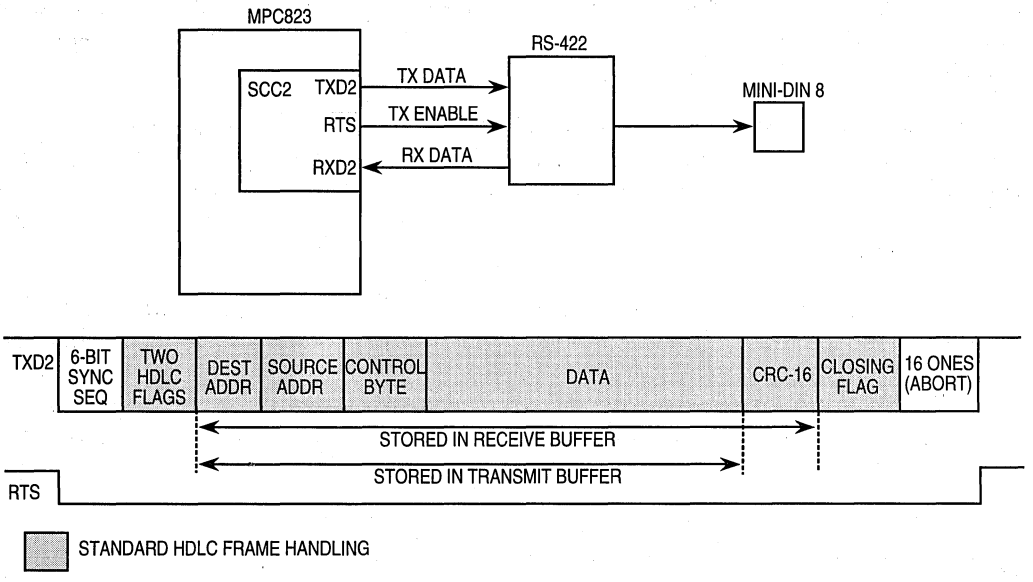


Figure 16-88. Connecting the MPC823 to AppleTalk

The 16× multiplier of a 3.686MHz clock can be generated from an external frequency source or from one of the baud rate generators if the resulting output frequency is close to a multiple of the 3.686MHz frequency. The MPC823 asserts the $\overline{\text{RTS}}$ signal throughout the duration of the frame so that $\overline{\text{RTS}}$ can be used to enable the RS-422 transceiver.

16.9.18.4 PROGRAMMING THE SCC2 IN APPLETALK MODE. You can implement the SCC2 AppleTalk controller by setting certain bits in the SCC2 HDLC controller. Otherwise, you should consult **Section 16.9.17 The HDLC Bus Controller** for detailed information about how to program the SCC2 HDLC controller. You can use the serial communication controller's GSMR, PSMR, or TODR to program the AppleTalk controller.

Use the general SCC2 mode high and low registers (GSMR_x) described in **Section 16.9.2 The General SCC2 Mode Registers** to program the AppleTalk controller with the following steps:

1. Set the MODE field in the GSMR_L to AppleTalk.
2. Set the ENT and ENR bits in the GSMR_L.
3. Set the DIAG field in the GSMR_L for normal operation, with the $\overline{CD}$ and $\overline{CTS}$ pins grounded or configured for parallel I/O. This causes $\overline{CD}$ and $\overline{CTS}$ to be internally asserted to the serial communication controller.
4. Set the RDCR and TDCR fields in the GSMR_L to a 16× clock.
5. Set the TENC and RENC fields in the GSMR_L to FM0.
6. Set the TEND bit in the GSMR_L to 0.
7. Set the TPP field in the GSMR_L to 11.
8. Set the TPL field to 000 to transmit the next frame with no synchronization sequence and to 001 to transmit the next frame with the LocalTalk synchronization sequence. For example, data frames do not require a preceding synchronization sequence. This field may be modified on-the-fly if the AppleTalk protocol is selected.
9. Set the TINV and RINV bits in the GSMR_L to zero.
10. Set the TSNC field in the GSMR_L to 10.
11. Set the EDGE field to 0.
12. Set the RTSM bit in the GSMR_H to 0.
13. Set all other bits to 0 or default.

Use the SCC2 protocol-specific mode register described in **Section 16.9.3 Protocol-Specific Mode Register** to program the AppleTalk controller with the following steps:

1. Set the NOF field to 0001 (binary) giving two flags before frames (one opening flag and one additional flag).
2. Set the CRC field to 16-bit CRC-CCITT.
3. Set the DRT bit to 1.
4. Set all other bits to 0 or default.

Use the transmit-on-demand register described in **Section 16.9.5 Transmit-on-Demand Register** to expedite a transmit frame by setting the TOD bit to 1.

16.9.18.5 SCC2 APPLETALK PROGRAMMING EXAMPLE. Except for the previously discussed register programming, use the example in **Section 16.9.16.14 SCC2 HDLC Programming Example #1**.

16.9.19 The SCC2 in Asynchronous HDLC Mode

Asynchronous HDLC is a frame-based protocol that uses HDLC framing techniques in conjunction with UART-type characters. This protocol is typically used as the physical layer for the point-to-point protocol (PPP) and the infra-red link access protocol (IRLAP). Even though asynchronous HDLC can be implemented in conjunction with the core, it is more efficient and less computationally intensive to allow the communication processor module to perform the framing and transparency functions. The SCC2 in ASYNC HDLC mode is also referred to as the SCC2 ASYNC HDLC controller.

16.9.19.1 FEATURES. The following list summarizes the main features of the SCC2 in asynchronous HDLC mode:

- Flexible data buffer structure that allows an entire frame or a section of one to be transmitted and received
- Separate interrupts for received frames and transmitted buffers
- Automatic CRC generation and checking
- Support for nonmultiplexed serial interface control signals
- Automatic generation of opening and closing flags
- Reception of frames with only one "shared" flag
- Automatic generation and stripping of transparency characters according to RFC 1549 using transmit and receive control character maps
- Programmable opening flag, closing flag, and control escape characters
- Automatic transmission of the abort sequence after the **STOP TRANSMIT** command is issued
- Automatic transmission of idle characters between frames

16.9.19.2 SCC2 ASYNC HDLC CHANNEL FRAME TRANSMISSION PROCESS. The SCC2 ASYNC HDLC controller, is designed to work with a minimum amount of intervention from the core and operates similar to the SCC2 in HDLC mode. When the core enables the transmitter and sets the R bit in the first transmit buffer descriptor, the SCC2 ASYNC HDLC controller fetches the data from memory and starts transmitting the frame. When the controller reaches the end of the current buffer descriptor, the CRC and closing flag are appended if the L bit in the TX buffer descriptor is set. If the CM bit is clear, the asynchronous HDLC transmitter writes the frame status bits into the buffer descriptor and clears the R bit. If the I bit is set, the controller sets the TXB bit in the SCCE–ASYNC HDLC register. Thus, the I bit can be used to generate an interrupt after each buffer, after a group of buffers, or after each complete frame has been transmitted.

If the CM bit in the TX buffer descriptor is set, the SCC2 ASYNC HDLC controller writes the signal unit status bits into the buffer descriptor after transmission but it does not clear the R bit. The SCC2 ASYNC HDLC controller then proceeds to the next TX buffer descriptor in the table. If it is not ready, it waits until it is ready. While the SCC2 ASYNC HDLC controller is transmitting data from the buffers, it automatically performs the transparency encoding specified by the protocol. This encoding is described in **Section 16.9.19.4 Transmitter Transparency Encoding**.

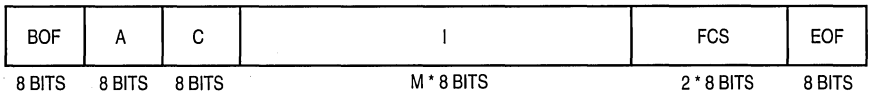


Figure 16-89. ASYNC HDLC Frame Structure

You should issue the **STOP TRANSMIT** command to rearrange the transmit queue before the communication processor module finishes transmitting all the buffers. This can be useful when transmitting expedited data prior to previously linked buffers or for error situations. When the SCC2 ASYNC HDLC controller receives the **STOP TRANSMIT** command, it stops transmitting and sends the abort sequence. It then transmits idle characters until the **RESTART TRANSMIT** command is given, at which point it resumes transmission with the next TX buffer descriptor.

16.9.19.3 SCC2 ASYNC HDLC CHANNEL FRAME RECEPTION PROCESS. The SCC2 ASYNC HDLC receiver is designed to work with a minimum amount of intervention from the core and can decode the transparency characters, check the CRC of the frame, and detect errors on the line and in the controller. When the core enables the receiver, the receiver waits for data to be present on the line. When the receiver detects a data byte of the incoming frame that was preceded by one or more opening flags, the SCC2 ASYNC HDLC controller fetches the next buffer descriptor and if the E bit is set it starts transferring the incoming frame into the buffer descriptor associated data buffer. When the data buffer is full, the SCC2 ASYNC HDLC controller clears the E bit in the buffer descriptor. If the incoming frame exceeds the length of the data buffer, the SCC2 ASYNC HDLC controller fetches the next buffer descriptor in the table and, if empty, continues transferring the rest of the frame into the associated data buffer.

During this process, the receiver automatically decodes the transparency character required of the SCC2 ASYNC HDLC protocol. This procedure is described in detail in **Section 16.9.19.5 Receiver Transparency Decoding**. When the frame ends, the controller checks the incoming CRC field and writes it to the data buffer. It then writes the length of the entire frame to the DATA LENGTH field of the last buffer descriptor. The SCC2 ASYNC HDLC controller sets the L bit, writes the frame status bits into the buffer descriptor, and clears the E bit if the CM bit is clear. It then sets the RXF bit in the SCCE–ASYNC HDLC register, which indicates that a frame has been received and is in memory. The SCC2 ASYNC HDLC controller then waits for the start of the next frame which may or may not have an opening flag.

16.9.19.4 TRANSMITTER TRANSPARENCY ENCODING. The SCC2 ASYNC HDLC controller maps characters according to the RFC 1549. It examines the outgoing data bytes and performs a transparency algorithm on a given byte if one of the following conditions apply:

- The byte is a flag (0x7E-PPP, 0xC0/0xC1-IrLAP)
- The byte is a control-escape character (0x7D)
- The byte has a value between 0x00 and 0x1F and the corresponding bit in the TX control character table is set

When a condition applies, a two-byte sequence is transmitted in place of the byte. The sequence consists of the control-escape character (0x7D) followed by the original byte exclusive-OR'ed with 0x20.

16.9.19.5 RECEIVER TRANSPARENCY DECODING. The SCC2 ASYNC HDLC controller maps characters according to the RFC 1549. To recover the original data, it examines the incoming data bytes and performs the transparency algorithm in the following ways:

- Discards any character that has its corresponding bit set in the RX control character map. This character is assumed to have been inserted in the character stream by an intermediate device and is not part of the originally transmitted frame.
- Reverses the transmission transparency sequence by discarding a received control-escape character (0x7D) and exclusive-OR'ing the following byte with 0x20 before performing the CRC calculation and writing the byte into memory.

Figure 16-90 illustrates the algorithm because there are some cases that are not addressed in RFC 1549.

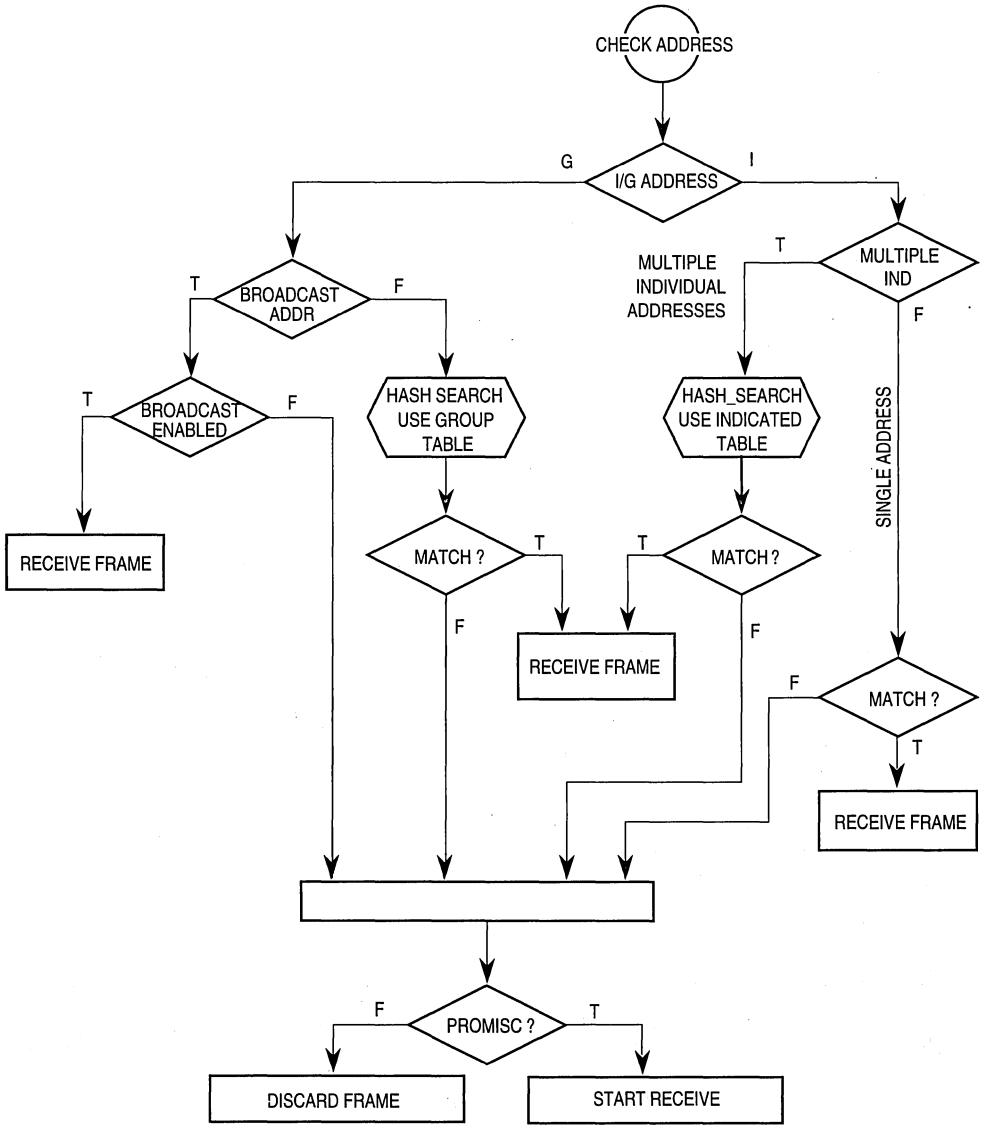


Figure 16-90. Reception Flowchart

16.9.19.6 EXCEPTIONS TO RFC 1549. The following beheviors do not conform to the RFC 1549.

- If an 0x7D is followed by a control character and the control character is not mapped, the control character itself is "modified" by the XOR process. The CRC check should catch this exception. This is a case where the transmitter control character table differs from the receiver.
- In addition to the abort sequence, frames are terminated by the following errors:
  - ❑ Carrier detect lost
  - ❑ Receiver overrun
  - ❑ Framing error
  - ❑ Break sequence
- If the invalid sequence is received, the first control escape character is discarded, and the second is unconditionally exclusive-OR'ed with 0x20. This sequence is stored in the buffer descriptor as 0x5D.

16.9.19.7 SCC2 ASYNC HDLC IMPLEMENTATION. The following behaviors represent the key aspects of the SCC2 ASYNC HDLC controller.

- Flag Sequence—When transmitting, the controller automatically generates the opening and closing flag for the frame. When receiving, the controller strips off the opening and closing flag before writing the frame to memory. It receives frames with only one "shared" flag between them and ignores multiple flags between frames.
- Address Field—The address field is neither generated nor examined by the microcode while transmitting or receiving. The address field of the frame must be included in the data buffer that the transmit buffer descriptor points to. Any address field compression, expansion, or checking must be performed by the core.
- Control Field—The control field is neither generated nor examined by the microcode while transmitting or receiving. The control field of the frame must be included in the data buffer that the transmit buffer descriptor points to. Any control field compression, expansion, or checking must be performed by the core.
- Frame Check Sequence—When transmitting, the frame check sequence (CRC) is automatically appended to the end of the frame before the closing flag is transmitted. The frame check sequence is generated on the original frame before the transparency characters, start/stop bits, or flags are added. The controller uses a 16-bit CRC-CCITT polynomial. When receiving, the frame check sequence is automatically checked. The frame check sequence is calculated after any transparency characters, start/stop bits, and flags are removed. The controller uses a 16-bit CRC-CCITT polynomial.
- Encoding—The SCC2 ASYNC HDLC controller only supports 8 data bits, one start bit, one stop bit, and no parity. This must be programmed in the PSMR–SCC2 ASYNC HDLC register so that bits 2 and 3 are set to 1 for proper operation.
- Time-Fill (Idling)—When transmitting, the SCC2 ASYNC HDLC controller transmits IDLE characters when no data is available for transmission. When receiving, the SCC2 ASYNC HDLC controller ignores IDLE characters.

16.9.19.8 SCC2 ASYNC HDLC PARAMETER RAM MEMORY MAP. When configured as an SCC2 ASYNC HDLC controller, the serial communication controller overlays the structure used in Table 16-24 with the SCC2 ASYNC HDLC parameters described in Table 16-28.

Table 16-28. SCC2 ASYNC HDLC Parameter RAM Memory Map

ADDRESS	NAME	WIDTH	DESCRIPTION
SCC2 Base+34	C_MASK	Word	CRC Constant
SCC2 Base+38	C_PRES	Word	CRC Preset
SCC2 Base+3C	BOF	Half-word	Beginning Of Flag Character
SCC2 Base+3E	EOF	Half-word	End Of Flag Character
SCC2 Base+40	ESC	Half-word	Control Escape Character
SCC2 Base+42	RES	Half-word	Reserved
SCC2 Base+44	RES	Half-word	Reserved
SCC2 Base+46	ZERO	Half-word	Reserved
SCC2 Base+48	RES	Half-word	Reserved
SCC2 Base+4A	RFTHR	Half-word	Received Frames Threshold
SCC2 Base+4C	RES	Half-word	Reserved
SCC2 Base+4E	RES	Half-word	Reserved
SCC2 Base+50	TXCTL_TBL	Word	TX Control Character Mapping Table
SCC2 Base+54	RXCTL_TBL	Word	RX Control Character Mapping Table
SCC2 Base+58	NOF	Half-word	Number of Opening Flags
SCC2 Base+5A	RES	Half-word	Reserved

NOTE: You are only responsible for initializing the items in bold.
SCC Base = (IMMR & 0xFFFF0000) + 0x3D00.

- C_MASK—This value should be initialized with 0x0000F0B8.
- C_PRES—This value should be initialized with 0x0000FFFF.
- BOF—This value should be initialized to the beginning of flag character (PPP is 0x7E and IRLAP is 0xC0).
- EOF—This value should be initialized to the end of flag character (PPP is 0x7E and IrLAP is 0xC1).
- ESC—This value should be initialized to the control escape character (PPP is 0x 7D and IrLAP is 0x7D).
- Reserved—These areas are temporary storage locations for the microcode. They should not be initialized or modified.
- ZERO—You must set this field to zero.

- RFTHR—The received frames threshold indicates how many frames are received before the RXF bit is set in the SCCE–ASYNC HDLC register.
- TXCTL_TBL—The transmit control character table stores the bit array used for the TX control character table. Each bit corresponds to a character that should be mapped according to RFC 1549. If the bit is set, the character corresponding to that bit is mapped and if the bit is not set, the corresponding character is not mapped. The transmit control character table should be initialized to zero for IrLAP.

TXCTL_TBL

BYTE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
VALUE	Ox1F	Ox1E	Ox1D	Ox1C	Ox1B	Ox1A	Ox19	Ox18	Ox17	Ox16	Ox15	Ox14	Ox13	Ox12	Ox11	Ox10
BYTE	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
VALUE	Ox9	Ox8	Ox7	Ox6	Ox5	Ox4	Ox3	Ox2	Ox1	Ox0	OxO5	OxO4	OxO3	OxO2	OxO1	OxOO

- RXCTL_TBL—The receive control character table stores the bit array used for the RX control character table. Each bit corresponds to a character that should be mapped according to RFC 1549. If the bit is set, the character corresponding to that bit is discarded if received and if the bit is not set, the corresponding character is received normally. The receive control character table should be initialized to zero for IrLAP.

RXCTL_TBL

BYTE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
VALUE	Ox1F	Ox1E	Ox1D	Ox1C	Ox1B	Ox1A	Ox19	Ox18	Ox17	Ox16	Ox15	Ox14	Ox13	Ox12	Ox11	Ox10
BYTE	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
VALUE	Ox9	Ox8	Ox7	Ox6	Ox5	Ox4	Ox3	Ox2	Ox1	Ox0	OxO5	OxO4	OxO3	OxO2	OxO1	OxOO

- NOF—This entry should be initialized to the number of opening flags to be transmitted at the beginning of a frame. A value of *n* corresponds to *n*+1 flags.

16.9.19.9 CONFIGURING THE SCC2 ASYNC HDLC PARAMETERS. The SCC2 ASYNC HDLC parameters can be configured as described in **Section 16.9 The Serial Communication Controller** through **Section 16.9.8 Handling Interrupts In the SCC2**, except for the changes in the following registers. When you are in asynchronous HDLC mode, some of the bits in the GSMR_x and general DSR have different meanings.

For the SCC2 in ASYNC HDLC mode, the general SCC2 mode high and low register (GSMR_x) bit descriptions remain the same, except for:

- RFW—RX FIFO Width (GSMR_H)

  0 = Should not be used.
  1 = Low-latency operation. The RX FIFO is 8 bits wide and the receive FIFO is one quarter its normal size (8 bytes). This allows data to be written to the data buffer each time a character is received, without forcing you to wait for 32 bits to be received. You must choose this configuration for character-oriented protocols like UART and asynchronous HDLC.

- TDCR—Transmit Divide Clock Rate (GSMR_L)

  These bits determine the divider rate of the transmitter. If the DPLL is not used, you must choose the 1× value. In asynchronous UART or HDLC mode, you must choose 8×, 16×, or 32×. You should program these bits to equal the RDCR field in most applications.

  00 = Do not use.
  01 = 8× clock mode (do not use for IrLAP).
  10 = 16× clock mode.
  11 = 32× clock mode (do not use for IrLAP).

- RDCR—Receive DPLL Clock Rate (GSNR_L)

  These bits determine the divider rate of the receive DPLL. If the DPLL is not used, you must choose the 1× value. In asynchronous UART or HDLC mode, you must choose 8×, 16×, or 32×. You should program these bits to equal the TDCR field in most applications.

  00 = Do not use.
  01 = 8× clock mode (do not use for IrLAP).
  10 = 16× clock mode.
  11 = 32× clock mode (do not use for IrLAP).

The data synchronization register (DSR) is reserved in asynchronous HDLC mode. It should be set to 0x7E7E.

16.9.19.10 SCC2 ASYNC HDLC COMMANDS. You can program the CPM command register (CPCR) with the following commands to transmit data. See **Section 16.2.7.1 CPM Command Register** for additional information. After the hardware or software is reset and the channel is enabled by the SCCM–ASYNC HDLC register, the channel is in transmit enable mode and starts polling the first buffer descriptor in the table every eight transmit bit times or immediately if the TOD bit of the TODR is set.

- **STOP TRANSMIT**—This command transmits the asynchronous HDLC abort sequence (PPP–0x7D ;0x7E, IrLAP–0x7D ;0xC1) and disables the transmission of data. If the ASYNC HDLC controller receives this command during frame transmission, the abort sequence is put into the FIFO and the transmitter does not try to send any more data from the current TX buffer descriptor. The controller also does not advance to the next TX buffer descriptor. You determine which buffer descriptor is terminated by examining the TBPTR entry in the SCC2 parameter RAM table. However, no new buffer descriptor is accessed for this channel.

Note: Unlike the other MPC823 protocols, the SCC2 ASYNC HDLC controller does not flush the FIFO because of the **STOP TRANSMIT** command. Up to 32 characters can be transmitted ahead of the abort sequence. However, this can be avoided by programming TFL to 1 in the GSMR_H register.

- **GRACEFUL STOP TRANSMIT**—This command is not supported by the SCC2 ASYNC HDLC controller.
- **RESTART TRANSMIT**—This command reenables the transmission of characters on the transmit channel and the SCC2 ASYNC HDLC controller expects it after a **STOP TRANSMIT** command or transmitter error. The controller continues transmitting from the first character in the current transmitter buffer descriptor in the channel's transmit buffer descriptor table.
- **INIT TX PARAMETERS**—This command initializes all the transmit parameters in this serial channel's parameter RAM to their reset state. It must be issued before the transmitter is enabled and should only be issued when the transmitter is disabled. The **INIT TX AND RX PARAMS** command can also be used to reset the transmit and receive parameters.

You can program the CPM command register (CPCR) with the following commands to receive data. After the hardware or software is reset and the channel is enabled by its SCCM–ASYNC HDLC register, the channel is in receive enable mode and uses the first buffer descriptor in the table.

- **ENTER HUNT MODE**—This command is used to force the SCC2 ASYNC HDLC controller to close the current RX buffer descriptor if it is being used and enter hunt mode. The controller continues receiving after it finds a frame preceded by one or more opening flags.
- **CLOSE RX BD**—This command is not supported by the SCC2 ASYNC HDLC controller.
- **INIT RX PARAMETERS**—This command initializes all the receive parameters in this serial channel's parameter RAM to their reset state and should only be issued when the receiver is disabled. The **INIT TX AND RX PARAMS** command can also be used to reset the receive and transmit parameters.

16.9.19.11 SCC2 ASYNC HDLC CONTROLLER ERRORS. The SCC2 ASYNC HDLC controller reports frame reception and transmission error conditions using the channel buffer descriptors and the SCC2 ASYNC HDLC event register. The following transmission error can be detected by the SCC2 ASYNC HDLC controller.

- CTS Lost During Frame Transmission Error—When this error occurs, the channel stops transmitting the buffer, closes it, sets the CT bit in the TX buffer descriptor and the TXE bit in the SCCE–ASYNC HDLC. The channel continues transmitting from the next TX buffer descriptor after the **RESTART TRANSMIT** command is issued.

The following reception errors can be detected by the SCC2 ASYNC HDLC controller.

- Overrun Error—The SCC2 ASYNC HDLC controller maintains an internal 8-byte FIFO when the serial communication controller receives data. A receive overrun occurs when the communication processor module is unable to keep up with the data rate or the SDMA channel is unable to write the received data to memory. The previous data byte and the frame status are lost. The controller closes the buffer with the OV bit in the buffer descriptor set and sets the RXF bit in the SCCE–ASYNC HDLC register. The receiver then searches for the next frame.
- CD Lost During Frame Reception Error—When this error occurs, the channel stops receiving frames, closes the buffer, and sets the CD bit in the buffer descriptor and the RXF bit in the SCCE–ASYNC HDLC register. This error has the highest priority. The rest of the frame is lost and other errors are not checked in that frame. The receiver then searches for the next frame once the $\overline{CD}$ signal is reasserted.
- Abort Sequence Error—This error occurs when the SCC2 ASYNC HDLC controller receives an abort sequence. At that time, the channel closes the buffer by setting the RX AB bit in the buffer descriptor and sets the RXF bit in the SCCE–ASYNC HDLC register. The CRC error status condition is not checked on aborted frames. If the abort sequence is received and no frame is currently being received, the next buffer descriptor is opened and then closed with the AB bit set in the buffer descriptor.

- CRC Error—When this error occurs, the channel writes the received cyclic redundancy check to the data buffer, closes the buffer, and sets the CR bit in the buffer descriptor and the RXF bit in the SCCE–ASYNC HDLC register. After receiving a signal unit with this error, the receiver gets ready to receive the next frame.
- Break Sequence Received Error—This error occurs when the UART receiver finds the first character of a break sequence. The channel closes the buffer by setting the RX BRK bit in the buffer descriptor and the RXF bit in the SCCE–ASYNC HDLC register. The CRC error status condition is not checked. The BRKSbit is set in the SCCE–ASYNC HDLC register when the first break of a sequence is found and the BRKE is set when an idle bit is received after a break sequence.

16.9.19.12 PROGRAMMING THE SCC2 ASYNC HDLC CONTROLLER.

16.9.19.12.1 SCC2 ASYNC HDLC Mode Register. When the SCC2 is in asynchronous HDLC mode, the 16-bit, memory-mapped, read/write protocol-specific mode register is referred to as the SCC2 ASYNC HDLC mode (PSMR–SCC2 ASYNC HDLC) register. It controls asynchronous HDLC mode-specific parameters. Since each protocol has specific requirements, the PSMR bits are different for each implementation.

PSMR–SCC2 ASYNC HDLC

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	FLC	RES	CHLN		RESERVED											
RESET	0	0	0		0											
R/W	R/W	R/W	R/W		R/W											
ADDR	(IMMR & 0xFFFF0000) + 0xA28															

FLC—Flow Control

0 = Normal operation.

1 = Asynchronous flow control. When the $\overline{CTS}$ pin is negated, the transmitter stops at the end of the current character. If $\overline{CTS}$ is negated past the middle of the current character, the next full character can be sent and transmission stops. When $\overline{CTS}$ is asserted once more, transmission continues where it left off and no $\overline{CTS}$ lost error is reported. No characters, except idles, are transmitted while $\overline{CTS}$ is negated.

Bits 1 and 4–15—Reserved

These bits are reserved and should be set to 0.

CHLN—Character Length

For asynchronous HDLC and IrLAP modes, these bits must be set to 1.

16.9.19.12.2 SCC2 ASYNC HDLC Receive Buffer Descriptor. The SCC2 ASYNC HDLC controller uses the receive (RX) buffer descriptor to report information about each buffer's received data. An example of the RX buffer descriptor process is illustrated in Figure 16-78. The first word of the RX buffer descriptor contains control and status bits. Bit 0 is set by the core when the buffer is available to the SCC2 ASYNC HDLC controller and it is cleared by the controller when the buffer is full.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	**E**	RES	**W**	**I**	L	F	**CM**	RES	BRK	BOF	RES		AB	CR	OV	CD
OFFSET + 2	DATA LENGTH															
OFFSET + 4	**RX DATA BUFFER POINTER**															
OFFSET + 6																

NOTE: You are only responsible for initializing the items in bold.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

E—Empty

0 = The data buffer associated with this buffer descriptor is filled with data or stops receiving because an error condition occurred. The core is free to examine or write to any fields of this RX buffer descriptor. The communication processor module does not use this buffer descriptor again as long as the E bit is zero.

1 = The data buffer associated with this buffer descriptor is empty or currently receiving data. This RX buffer descriptor and its associated receive buffer are owned by the communication processor module. Once the E bit is set, the core should not write any fields of this RX buffer descriptor.

Bits 1, 7, and 10–11—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the RX buffer descriptor table.

1 = This is the last buffer descriptor in the RX buffer descriptor table. After this buffer is used, the communication processor module receives incoming data into the first buffer descriptor that RBASE points to in the table. The number of RX buffer descriptors in this table are programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0 = The RXB bit in the SCCE–ASYNC HDLC register is not set after this buffer is used, but RXF operation is unaffected.
1 = The RXB or RXF bit in the SCCE–ASYNC HDLC register is set when this buffer is used by the SCC2 ASYNC HDLC controller.

L—Last in Frame

This bit is set by the SCC2 ASYNC HDLC controller when this buffer is the last in a frame. If a closing flag or error is received, one or more of the BRK, CD, OV, and AB bits are set. The SCC2 ASYNC HDLC controller writes the number of frame octets to the DATA LENGTH field.

0 = This buffer is not the last one in a frame.
1 = This buffer is the last one in a frame.

F—First in Frame

This bit is set by the SCC2 ASYNC HDLC controller when this buffer is the first in a frame.

0 = The buffer is not the first one in a frame.
1 = The buffer is the first one in a frame.

CM—Continuous Mode

0 = Normal operation.
1 = The E bit is not cleared by the communication processor module after this buffer descriptor is closed, thus allowing the associated data buffer to be automatically overwritten next time the buffer descriptor is accessed by the communication processor module. However, the E bit is cleared if an error other than CRC occurs during reception, regardless of how the CM bit is set.

BRK—Break Character Received

This bit indicates that the current frame is closed when a break character is received.

BOF—Beginning of Frame Encountered

This bit indicates that the current frame is closed when a BOF character is received instead of the expected EOF.

AB—RX Abort Sequence

This bit indicates that an asynchronous HDLC abort sequence or framing error is received to terminate this frame.

CR—RX CRC Error

This bit indicates that this frame contains a CRC error. The received CRC bytes are always written to the receive buffer.

OV—Overrun

This bit indicates that a receiver overrun has occurred during frame reception.

CD—Carrier Detect Lost

This bit indicates that the carrier detect signal is negated during frame reception.

DATA LENGTH

These bits represent the number of octets the communication processor module writes into this buffer descriptor data buffer once the buffer descriptor is closed. When this buffer descriptor is the last one in the frame, DATA LENGTH contains the total number of frame octets. The actual amount of memory allocated for this buffer should be greater than or equal to the contents of the MRBLR.

Note: If the received frame has a length that is an exact multiple of the MRBLR, the buffer descriptor with the L bit set does not actually have any characters in it and the DATA LENGTH field contains a value equal to the sum of the DATA LENGTH fields of the other buffer descriptors in the frame.

RX DATA BUFFER POINTER

These bits always point to the first location of the associated data buffer and can reside in internal or external memory.

16.9.19.12.3 SCC2 ASYNC HDLC Transmit Buffer Descriptor. Data is sent to the SCC2 ASYNC HDLC controller for transmission on an SCC2 channel by arranging it in buffers referenced by the channel transmit (TX) buffer descriptor table. Using the buffer descriptors, the SCC2 ASYNC HDLC controller confirms transmission or indicates error conditions so that the core will know the buffers have been serviced.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	R	RES	W	I	L	RES	CM	RES								CT
OFFSET + 2	DATA LENGTH															
OFFSET + 4	TX DATA BUFFER POINTER															
OFFSET + 6																

NOTE: You are only responsible for initializing the items in bold.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

R—Ready

0 = The data buffer associated with this buffer descriptor is not ready for transmission. You are free to manipulate this buffer descriptor or its associated data buffer. The communication processor module clears this bit after the buffer is transmitted or after an error condition is encountered.

1 = The data buffer, which you have prepared for transmission, is not transmitted yet or is currently being transmitted. You cannot write any fields of this buffer descriptor once this bit is set.

Bits 1, 5, and 7–11—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the TX buffer descriptor table.

1 = This is the last buffer descriptor in the TX buffer descriptor table. After this buffer is used, the communication processor module receives incoming data into the first buffer descriptor that TBASE points to in the table. The number of TX buffer descriptors in this table are programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0= The TXB bit in the SCCE–ASYNC HDLC register is not set after this buffer is used.

1= The TXB bit in the SCCE–ASYNC HDLC register is set when this buffer is transmitted by the SCC2 ASYNC HDLC controller.

L—Last

0= This is not the last buffer in the current frame.

1= This is the last buffer in the current frame. The proper CRC and closing FLAG are transmitted after the last byte is transmitted.

CM—Continuous Mode

0 = Normal operation.

1 = The R bit is not cleared by the communication processor module after this buffer descriptor is closed, thus allowing the associated data buffer to be automatically retransmitted the next time the communication processor module accesses this buffer descriptor. However, the R bit is cleared if an error occurs during transmission, regardless of how the CM bit is set.

CT—CTS Lost

In NMSI mode, this bit indicates that $\overline{CTS}$ is lost during frame transmission. If data from more than one buffer is currently in the FIFO when this error occurs, this bit is set in the currently open TX buffer descriptor. These bits are written by the SCC2 ASYNC HDLC controller after it finishes transmitting the associated data buffer.

DATA LENGTH

These bits represent the number of bytes the SCC2 ASYNC HDLC controller should transmit from this buffer descriptor data buffer. It is never modified by the communication processor module. The value of this field must be greater than zero. These bits are written by the SCC2 ASYNC HDLC controller after it finishes transmitting the associated data buffer.

TX DATA BUFFER POINTER

These bits contain the address of the associated data buffer, can be even or odd, and can reside in internal or external memory. This value is never modified by the communication processor module. These bits are written by the SCC2 ASYNC HDLC controller after it finishes transmitting the associated data buffer.

16.9.19.12.4 SCC2 ASYNC HDLC Event Register. When the SCC2 is in asynchronous HDLC mode, the 16-bit memory-mapped SCC2 event register is referred to as the SCC2 asynchronous HDLC event (SCCE–ASYNC HDLC) register. Since each protocol has specific requirements, the SCCE bits are different for each implementation. This register is used to generate interrupts and report events recognized by the SCC2 ASYNC HDLC channel. When an event is recognized, the SCC2 ASYNC HDLC controller sets the corresponding bit in the SCCE–ASYNC HDLC register. Interrupts generated by this register can be masked by the SCCM–ASYNC HDLC register.

A bit is cleared by writing a 1 (writing a zero has no effect) and more than one bit can be cleared at a time. However, all unmasked bits must be cleared before the communication processor module clears the internal interrupt request. This register is cleared at reset and can be read at any time.

SCCE–ASYNC HDLC

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED			GLr	GLt	RESERVED		IDL	RES	BRKE	BRKS	TXE	RXF	BSY	TXB	RXB
RESET	0			0	0	0		0	0	0	0	0	0	0	0	0
ADDR	(IMMR & 0xFFFF0000) + 0xA30															

Bits 0–2, 5–6, and 8—Reserved

These bits are reserved and should be set to 0.

GLR—Glitch on RX

This bit indicates that the serial communication controller has found a glitch on the receive clock.

GLT—Glitch on TX

This bit indicates that the serial communication controller has found a glitch on the transmit clock.

IDL—Idle Sequence Status Changed

This bit indicates that a change in the status of the serial line has occurred. The real-time status of the line can be read in the SCCS–ASYNC HDLC register.

TXE—TX Error

This bit indicates that an error has occurred on the transmitter channel.

BRKe—Break End

This bit indicates that the end of a break sequence has been found. This indication is set when one idle bit is received after a break sequence.

BRKS—Break Start

This bit indicates that a break character has been received. This is the first break of a break sequence. You will not receive multiple BRKS events if a long break sequence is detected.

RXF—RX Frame

This bit indicates that the SCC2 ASYNC HDLC channel has received a complete frame. This bit is set no sooner than two bit times after the last bit of the closing flag is received.

BSY—Busy Condition

This bit indicates that a frame has been received and discarded due to a lack of buffers.

TXB—Transmit Buffer

This bit indicates that a buffer with its I bit set has been transmitted on the SCC2 ASYNC HDLC channel. This bit is set no sooner than when the last bit of the closing flag begins its transmission if the buffer is the last one in the frame. Otherwise, this bit is set after the last byte of the buffer is written to the transmit FIFO.

RXB—RX Buffer

This bit indicates that a buffer, that is not the last in the frame with its I bit set, has been received over the SCC2 ASYNC HDLC channel.

16.9.19.12.5 Differences Between HDLC and ASYNC HDLC. There are four basic differences in the way the HDLC and ASYNC HDLC modes operate and they are as follows:

- There is no maximum received frame length counter in the ASYNC HDLC controller. Therefore, the controller receives all characters between opening and closing flags and there is no way to stop the controller from writing to memory. This in no way affects the number of bytes received into a specific buffer descriptor. It just means that a frame is received into memory in its entirety.
- If an error causes a frame to stop being received, the character being received at the moment the error occurred is not written into memory. For example, if a $\overline{CD}$ lost error occurred, the frame is closed and the partial character is not written to memory. Thus, the octet count only reflects the number of bytes written to memory.
- The automatic error counters in HDLC mode have not been implemented in asynchronous HDLC mode.
- Noisy characters (those whose three samples are not the same) are not accounted for in the ASYNC HDLC controller. It is assumed that the CRC catches any data integrity problems.

Note: The **GRACEFUL STOP TRANSMIT** command is not supported by the ASYNC HDLC controller.

16.9.19.12.6 SCC2 ASYNC HDLC Programming Guide. The following is an initialization sequence guide for the SCC2 in asynchronous HDLC mode.

1. Initialize the SDCR.
2. In NMSI mode, configure the port A pins to enable RXD2 and TXD2.
3. Configure a baud rate generator to generate appropriate channel clocking frequency.
4. Program the SICR to route the baud rate generator clocking to the serial communication controller, which is running asynchronous HDLC.
5. Select whether the channel is using the time-slot assigner or the NMSI pins in the SICR.
6. Write RBASE and TBASE in the SCC2 parameter RAM to point to the first RX buffer descriptor and TX buffer descriptor.
7. Issue the **INIT RX AND TX PARAMS** command for the serial communication controller.
8. Program the RFCR and TFCR.
9. Write the MRBLR with the maximum receiver buffer size.
10. Write C_MASK and C_PRES with the standard values.
11. Write the ZERO register to 0x0000.
12. Program the RFTHR to the number of frames that should be received before an interrupt is generated.

13. Program the TX and RX control character tables.
14. Initialize all RX buffer descriptors.
15. Initialize all TX buffer descriptors.
16. Clear the SCCE–ASYNC HDLC register by writing 0xFFFF to it.
17. Program the SCCM–ASYNC HDLC register with the proper mask to allow all preferred interrupts.
18. Program the GSMR_H register.
19. Program the GSMR_L register to asynchronous HDLC mode, but do not turn on the transmitter or receiver.
20. Set the PSMR–SCC2 ASYNC HDLC appropriately.
21. Turn on the transmitter and receiver in the GSMR_L register by setting the ENT and ENR bits.

16.9.20 The SCC2 in IrDA Mode

IrDA is a family of specifications intended to facilitate the interconnection of computers and peripherals using a directed half duplex serial infrared physical communications medium. The infra-red data association (IrDA) physical layer standard version 1.1 specifies three modes of operation, each one with a distinct modulation scheme and signaling rate.

- Low speed—2.4Kb/s to 115.2Kb/s
- Middle speed—0.576Mb/s or 1.152Mb/s
- High speed—4Mb/s

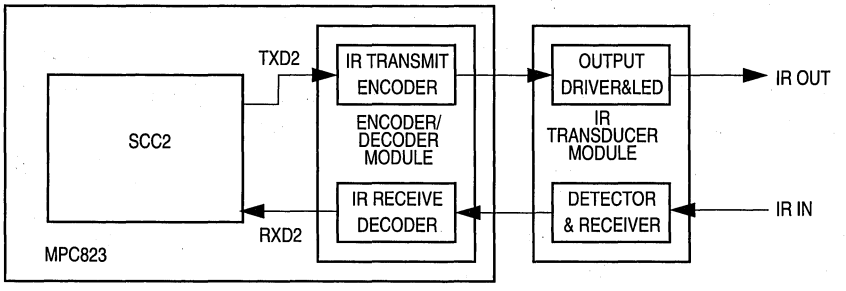


Figure 16-91. Serial IrDA Link

16.9.20.1 LOW-SPEED IRDA PROTOCOL. The low-speed IrDA protocol consists of a data link layer and a physical layer.

- The data link layer is based on a preexisting ASYNC HDLC protocol standard. See **Section 16.9.19 The SCC2 in Asynchronous HDLC Mode** for more details. 0xC0 is used as a start flag and 0xC1 is used as an end flag. Each character is compromised out of a start bit, 8 data bits, no parity bit and ending with a stop bit, as shown in Figure 16-92.
- The physical layer defines the electrical parameters of the signals between the encoder/decoder module and the infra-red transducer module. The signal waveform is shown in Figure 16-92. For all signaling rates up to and including 115.2Kb/s, the minimum pulse duration is the same—$^{3}/_{16}$ of the bit duration for the 115.2Kb/s signal (minus a protocol-defined tolerance). The maximum pulse duration is $^{3}/_{16}$ of the bit duration (plus a protocol-defined tolerance).

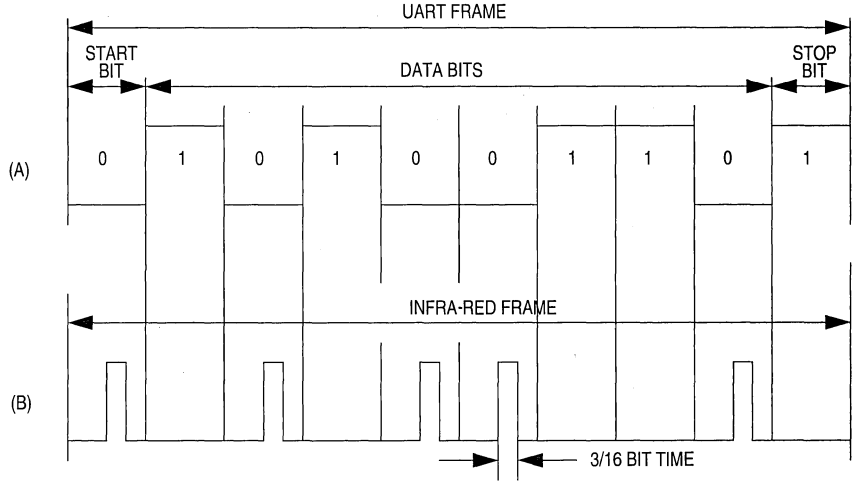


Figure 16-92. Low-Speed IrDA Data Format

16.9.20.2 MIDDLE-SPEED IRDA PROTOCOL. The middle-speed IrDA protocol consists of a data link layer and a physical layer.

- The data link layer is derived from the preexisting synchronous HDLC protocol standard. The frame format follows the standard HDLC format except that it requires two start flags. The frame consists of two start flags, an address field, a control field, an information field, a frame check sequence field and minimum of one ending flag. 0x7E is used for the start flag as well for the end flag.

START FLAG	START FLAG	ADDRESS	CONTROL	INFORMATION	CRC	END FLAG
1 BYTE	1 BYTE	1 BYTE	1 BYTE	N BYTES	2 BYTES	1 BYTE

Figure 16-93. Middle Speed Packet Format

- The physical layer defines the electrical parameters of the signals between the encoder/decoder module and the infra-red transducer module. The signal waveform is shown in Figure 16-94. For 0.576 and 1.152Mb/s, the minimum and maximum pulse duration are the nominal $^1/_4$ of the bit duration (plus or minus the protocol-defined tolerance).

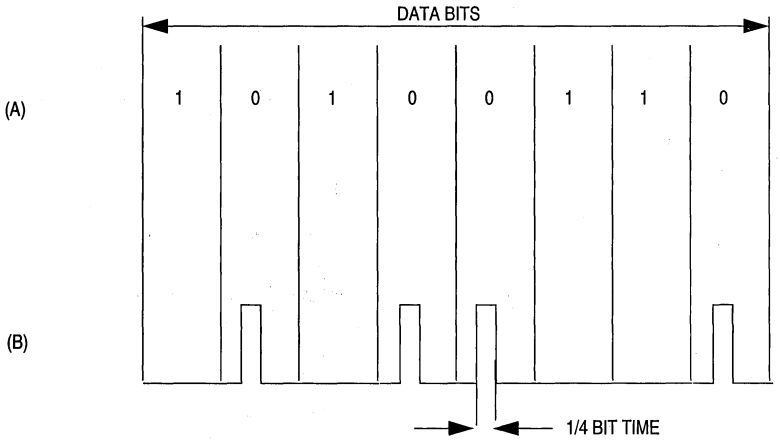


Figure 16-94. Middle-Speed IrDA Data Format

16.9.20.3 HIGH-SPEED IRDA PROTOCOL. The high-speed IrDA protocol is derived from the preexisting SCC2 Transparent protocol standard.

16.9.20.3.1 4PPM Data Encoding. Pulse position modulation (PPM) encoding is achieved by defining a data symbol duration (Dt) and subsequently subdividing Dt into a set of equal time slices called "chips". In PPM schemes, each chip position within a data symbol represents one of the possible bit combinations. Each chip has a duration of Ct given by Ct = Dt/Base.

In this formula *base* refers to the number of pulse positions or chips in each data symbol. The base for high-speed IrDA protocol is defined as four and the resulting modulation scheme is four-pulse position modulation (4PPM). The data rates of a IrDA PPM system are defined as 4.0Mb/s.

The resulting values for Ct and Dt are as follows:

- Dt = 500 ns
- Ct = 125 ns

The figure below illustrates a data symbol field and its enclosed chip durations for a 4PPM scheme.

Figure 16-95. One Complete Symbol

Because there are four unique chip positions within each symbol in 4PPM, four independent symbols exist in which only one chip is logically a "one" while other chips are logically a "zero". These four unique symbols are the only legal data symbols (DD) allowed in 4PPM. Each data symbol represents two bits of payload data, or a single data bit pair (DBP), so that a byte of payload data can be represented by four data symbols in sequence. The following table defines the chip pattern representation of the four unique data symbols defined for 4PPM.

DATA BIT PAIR	4PPM DATA SYMBOL
00	1000
01	0100
10	0010
11	0001

Logical "1" represents a chip duration when the transmitting LED is emitting light, while logical "0" represents a chip duration when the LED is off. Data encoding for transmission is done LSB first. The 4PPM data encoding defines only the legal encoded payload data symbols. All other 4-chip combinations are by definition illegal symbols for encoded payload data. Some of these illegal symbols are used in the definition of the packet envelope (preamble, start flag, stop flag) because they are unambiguously not data.

16.9.20.3.2 Data Link Layer. The data link layer protocol defines the following packet format.

PA	STA	LINK LAYER FRAME	STO

Figure 16-96. High-Speed Packet Format

The preamble (PA) field is used by the receiver to establish bit synchronization. The PA field consists of exactly sixteen repeated transmissions of the following stream of symbols.

1000	0000	1010	1000

Figure 16-97. Preamble Field Symbol Format

On the receive side, the PA field does not need to be valid in the received packet. After the PA field, the receiver starts searching for the start flag (STA) to establish symbol synchronization. After the start flag is received, the receiver can begin interpreting the data symbols in the link layer frame. The start flag consists of exactly one transmission of the following stream of symbols.

0000	1100	0000	1100	0110	0000	0110	0000

Figure 16-98. Start Flag Symbol Format

The link layer frame generally consists of the address, control, data and CRC32 fields. The IrDA transmitter decodes the packet bits into 4PPM format. The 4PPM encoding will be described later. The receiver is responsible to decode the incoming data frame into the regular bit format and to deliver it to the software. The receiver continues to receive and interpret data until the stop flag (STO) is recognized. A stop flag indicates the end of frame. The stop flag consists of exactly one transmission of the following stream of symbols.

0000	1100	0000	1100	0000	1100	0000	1100

Figure 16-99. Stop Flag Symbol Format

The physical layer defines the electrical parameters of the signals between the encoder/decoder module and the IR transducer module. All frame envelope patterns—PA, STA, and STO—are transmitted as is. The link layer frame bits are encoded before transmission. Each two bits encoded into four chips according to the 4PPM scheme.

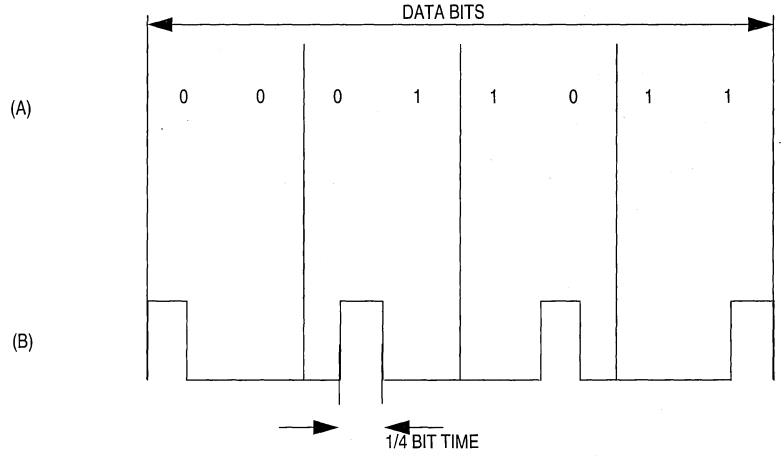


Figure 16-100. High-Speed IrDA Data Format

16.9.20.4 SERIAL INFRA-RED INTERACTION PULSES. To guarantee nondisruptive coexistence with slower (a maximum of 115.2Kb/s) systems, once a high-speed (above 115.2Kb/s) connection has been established the high-speed system must emit a serial infra-red interaction pulse (SIP) at least once every 500ms. This continues for the duration of the connection to quiet slower systems that might interfere with the link. The pulse is illustrated in Figure 16-101.

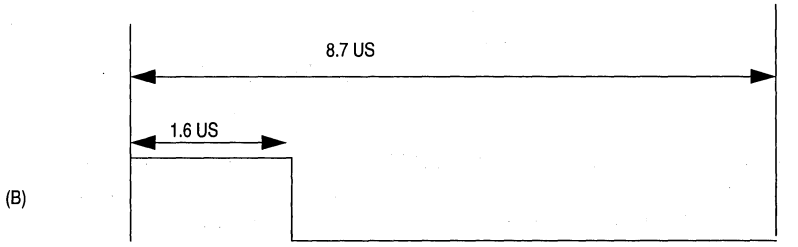


Figure 16-101. Serial Infra-Red Interaction Pulse Waveform

The serial infra-red interaction pulse can be generated in two ways:

- By the software when you set the GS bit in IRSIP register.
- By the Timer 2 expiration when you set the TS bit in the IRSIP register.

You are responsible for creating the appropriate SIP waveform by writing the proper values to the SLL and SHL fields of the IRSIP register.

16.9.20.5 PROGRAMMING MODEL.

16.9.20.5.1 SCC2 Infra-Red Mode Register. The SCC2 serial infra-red mode (IRMODE) register controls the infra-red operation mode (low-, middle-, or high speed), the number of preambles, the loop mode, the full-duplex operation, and the DPLL clock rate.

IRMODE

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PA				RESERVED		RXP	TXP	DCR		FD	LOOP	RES	MOD		EN
RESET	0				0		0	0	0		0	0	0	0		0
R/W	R/W				R/W		R/W	R/W	R/W		R/W	R/W	R/W	R/W		R/W
ADDR	(IMMR & 0xFFF0000) + 0xA38															

PA—Number of Preambles

This field determines the number of preambles in a high-speed transmitter. It is valid only in high-speed mode.

0000 = 16 preambles.
0001 = 1 preamble.
•
•
•
1111 = 15 preambles.

RXP—RX Polarity

This bit determines the polarity of the received signal.

0 = Active high polarity. An active high pulse is decoded as 0.
1 = Active low polarity. An active low pulse is decoded as 0.

TXP—TX Polarity

This bit determines the polarity of the transmitted signal.

0 = Active high polarity. An active high pulse is encoded as 0.
1 = Active low polarity. An active low pulse is encoded as 0.

DCR—IrDA DPLL Clock Ratio

This field determines the clock ratio between the IrDA DPLL clock and the bit rate clock. This field is valid only in high-speed mode.

00 = 12x bit rate clock.
01 = Reserved.
1x = Reserved.

FD—Full Duplex

0 = Data reception is disabled during the transmission process.
1 = Transmission and reception of data in parallel are enabled.

LOOP—Loop Mode

When set, this bit selects the local loopback operation. The transmitter output is internally connected to the receiver input. The receiver and transmitter operate normally, except that the received data is ignored.

0 = Normal operation.
1 = The IrDA is in loopback mode.

Note: The FD bit should be set in loopback mode.

M —Infra-Red Mode

Mode of operation.

00 = Low-speed mode (up to and including 115.2kb/s).
01 = Middle-speed (0.576Mb/s or 1.152Mb/s).
10 = High-speed (4.0Mb/s).
11 = Reserved.

EN—Enable IrDA

This bit enables IrDA decoder/encoder operation.

0 = IrDA is disabled.
1 = IrDA is enabled.

Note: Changing the EN bit value is allowed only when the SCC2 is off (after the ENR and ENT bits in the GSMR_L are cleared).

16.9.20.5.2 SCC2 Infra-Red Serial Interaction Pulse Control Register. The SCC2 infra-red serial interaction pulse control (IRSIP) register controls the initiation of the serial infrared interaction pulse.

IRSIP

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RES	GS	TS	RES	SHL					SLL						
RESET	0	0	0	0	0					0						
R/W	R/W	R/W	R/W	R/W	R/W					R/W						
ADDR	(IMMR & 0xFFF0000) + 0xA3A															

GS—Generate Serial Infra-Red Interaction Pulse

0 = Writing zero to this bit has no effect.
1 = Setting this bit generates a serial infra-red interaction pulse, which occurs only when the channel is idle. This bit is immediately reset by the IrDA controller.

Note: Reading the GS bit always returns a zero.

TS—Timer Set

0 = The Timer 2 status has no effect on the serial infra-red interaction pulse.
1 = The expiration of Timer 2 triggers the generation of a serial infra-red interaction pulse.

SHL—Serial Infra-Red Interaction Pulse High-Level Length

Program this field with the width of the SIP assertion part (in bit rate clock units).

SLL—Serial Infra-Red Interaction Pulse Low-Level Length

Program this field with the width of the SIP negation part (in bit rate clock units).

16.9.20.5.3 Low-Speed IrDA Programming Guide. Low-speed infra-red programming is very similar to SCC2 ASYNC HDLC programming. The only difference is the value of EOF and BOF in the SCC2 parameter RAM and the programming of the IRMODE register. Use the following initialization sequence for low-speed infra-red.

1. Initialize the SDCR with the appropriate arbitration ID.
2. Configure the port A and port C pins to enable $\overline{\text{RXD2}}$ and $\overline{\text{TXD2}}$. This assumes you are using NMSI mode. If not, appropriately configure the time-slot assigner and pins.
3. Configure a baud rate generator to generate the appropriate channel clocking frequency.
4. Program the SICR to route the baud rate generator clocking to the serial communication controller running SCC2 ASYNC-HDLC.
5. Select whether the channel is using the time-slot assigner or the NMSI pins in the SICR.
6. Write RBASE and TBASE in the SCC2 parameter RAM to point to the first RX buffer descriptor and the first TX buffer descriptor.
7. Issue the **INIT RX AND TX PARAMS** command to the serial communication controller.
8. Program RFCR and TFCR.
9. Write MRBLR with the maximum receive buffer size.
10. Write C_MASK and C_PRES with the standard values.
11. Write 0xC0 to BOF, 0xC1 to EOF and 0x7D to ESC.
12. Write 0 to the ZERO register in the parameter RAM.
13. Program the RFTHR to the number of frames that should be received before an interrupt is generated.
14. Program the TX and RX control character tables.
15. Initialize all RX buffer descriptors.
16. Initialize all TX buffer descriptors.
17. Clear the SCCE–ASYNC HDLC register by writing 0xFFFF to it.
18. Program the SCCM–ASYNC HDLC register with the proper mask to allow all desired interrupts.
19. Program the GSMR_H.
20. Program the MODE field of the GSMR_L to SCC2 ASYNC HDLC mode, but do not turn on the transmitter or receiver.
21. Write 0x0001 to the IRMODE register to enable low-speed infra-red.
22. Set the PSMR_SCC2 ASYNC HDLC appropriately.
23. Turn on the transmitter and receiver in the GSMR_L by setting the ENT and ENR bits.

16.9.20.5.4 Middle-Speed IrDA Programming Example. Middle-speed infra-red programming is very similar to SCC2 synchronous HDLC programming. The parameter RAM programming and the RX and TX buffer descriptors are the same as in the SCC2 HDLC. All of the SCC2 synchronous registers and the infra-red registers must be initialized. The following list is an initialization sequence for a middle-speed infra-red channel assuming that an external clock is provided. The CLK3 pin is used for the infra-red receiver and transmitter.

1. Configure the port A pins to enable the TXD2 and RXD2 pins. Write PAPAR bits 13 and 12 with ones. Write PADIR bits 13 and 12 with zeros. Write PAODR bits 13 and 12 with zeros.
2. Configure port A to enable the CLK3 pin. Write PAPAR bit 5 with a one. Write PADIR bit 5 with a zero.
3. Connect the CLK3 pin to the SCC2 using the serial interface. Write the R2CS field in SICR to 110. Write the T2CS bits in the SICR to 110.
4. Connect the SCC2 to the NMSI (its own set of pins). Clear the SC2 bit in the SICR.
5. Write the SDCR with the appropriate arbitration ID.
6. Write RBASE and TBASE in the SCC2 parameter RAM to point to the RX buffer descriptor and TX buffer descriptor in the dual-port RAM. Assuming one RX buffer descriptor at the beginning of dual-port RAM and one TX buffer descriptor following that RX buffer descriptor, write RBASE with 0x2000 and TBASE with 0x2008.
7. Program the CPCR to execute the **INIT RX AND TX PARAMS** command for the serial communication controller.
8. Write RFCR with 0x18 and TFCR with 0x18 for normal operation.
9. Write MRBLR with the maximum number of bytes per receive buffer. For this case, assume 256 bytes, so MRBLR = 0x0100. The value 256 was chosen to allow an entire receive frame to fit into one receive buffer.
10. Write CRC_P with 0xFFFFFFFF to comply with 16-bit CRC32.
11. Write CRC_C with 0xDEBB20E3 to comply with 16-bit CRC32.
12. Clear DISFC, CRCEC, ABTSC, NMARC, and RETRC for the sake of clarity.
13. Write MFLR with 0x0100 to make the maximum frame size 256 bytes.
14. Write RFTHR with 0x0001 to allow interrupts after each frame.
15. Write HMASK with 0x0000 to allow all addresses to be recognized.
16. Clear HADDR1, HADDR2, HADDR3, and HADDR4 for clarity.
17. Initialize the RX buffer descriptor. Assume the RX data buffer is at 0x00001000 in main memory. Write 0xB000 to RX_BD_Status. Write 0x0000 to RX_BD_Length (not required-done for instructional purposes only). Write 0x00001000 to RX_BD_Pointer.
18. Initialize the TX buffer descriptor. Assume the TX data frame is at 0x00002000 in main memory and contains five 8-bit characters. Write 0xBC00 to TX_BD_Status. Write 0x0005 to TX_BD_Length. Write 0x00002000 to TX_BD_Pointer.
19. Write 0xFFFF to the SCCE–HDLC to clear any previous events.

20. Write 0x001A to the SCCM–HDLC to enable the TXE, RXF, and TXB interrupts.
21. Write 0x20000000 to the CIMR to allow the SCC2 to generate a system interrupt. The CICR should also be initialized.
22. Write 0x00000000 to the MODE field of the GSMR_H to enable normal behavior of the $\overline{CTS}$ and $\overline{CD}$ pins and idles between frames (as opposed to flags).
23. Write 0x00028800 to the MODE field of the GSMR_L to configure the $\overline{CTS}$ and $\overline{CD}$ pins to automatically control transmission, reception (DIAG field), and HDLC mode. Normal operation of the transmit clock is selected and the TCI bit is cleared. The TDCR and the RDCR must be configured to 16x clock mode and the receiver decoding method must be NRZI. Notice that the transmitter (ENT) and receiver (ENR) have not been enabled. If you want inverted infra-red operation, set the RINV and TINV bits in the GSMR_L.
24. Set the PSMR–HDLC to 0x1000 to configure two opening and one closing flag, 16-bit CCITT-CRC, and prevention of multiple frames in the FIFO.
25. Write 0x0108 for a 1.152Mb/s infra-red rate or 0x0084 for a 0.576Mb/s infra-red rate to the IRSIP register. When working with Timer 2 as the SIP trigger, the values should be 0x2108 for a 1.153Mb/s infra-red or 0x2084 for a 0.572Mb/s.
26. Write 0x0003 to the IRMODE register to enable the infra-red and to set the mode of operation to middle-speed.
27. Program the TMR2 register when working with Timer 2 as the SIP trigger.
28. Write 0x00028830 to GSMR_L to enable the SCC2 transmitter and receiver. This additional write ensures that the ENT and ENR bits will be enabled last.

Note: After 5 bytes and CRC have been transmitted, the TX buffer descriptor is automatically closed. Once a complete frame is received, the RX buffer descriptor is closed. Any data received after 256 bytes or a single frame causes a busy (out-of-buffers) condition since only one RX buffer descriptor is prepared.

16.9.20.5.5 High-Speed IrDA Programming Example. High-speed infra-red programming is very similar to SCC2 Transparent programming. The parameter RAM programming and the RX buffer descriptor and TX buffer descriptor are the same as in the SCC2 transparent mode, which is described in **Section 16.9.21 The SCC2 in Transparent Mode**. The SCC2 and infra-red registers must be initialized. The following list is an initialization sequence for a high-speed infra-red channel. The transmitter and receiver are both enabled. Both transmit and receive clocks are provided externally to MPC823 using the CLK3 pin.

1. Configure the port A pins to enable the TXD2 and RXD2 pins. Write PAPAR bits 13 and 12 with ones. Write PADIR bits 13 and 12 with zeros. Write PAODR bits 13 and 12 with zeros.
2. Configure port A to enable the CLK3 pin. Write PAPAR bit 5 with a one. Write PADIR bit 5 with a zero.
3. Connect the CLK3 pin to SCC2 using the serial interface. Write the R2CS field in the SICR to 110. Write the T2CS field in the SICR to 110.
4. Connect the SCC2 to the NMSI (its own set of pins). Clear the SC2 bit in the SICR.
5. Write the SDCR with the appropriate arbitration ID.
6. Write RBASE and TBASE in the SCC2 parameter RAM to point to the RX buffer descriptor and TX buffer descriptor in the dual-port RAM. Assuming one RX buffer descriptor at the beginning of dual-port RAM, and one TX buffer descriptor following that RX buffer descriptor, write RBASE with 0x2000 and TBASE with 0x2008.
7. Program the CPCR to execute the **INIT RX AND TX PARAMS** command for the serial communication controller.
8. Write RFCR with 0x18 and TFCR with 0x18 for normal operation.
9. Write MRBLR with the maximum number of bytes per receive buffer. For this case, assume 16 bytes, so MRBLR = 0x0010.
10. Write 0xFFFFFFFF to CRC_P for 32-bit CRC-CCITT (CRC32). For details, see **Section 16.9.21.5 SCC2 Transparent Parameter RAM Memory Map**.
11. Write 0xDEBB20E3 to CRC_C for 32-bit CRC-CCITT (CRC32).
12. Initialize the RX buffer descriptor. Assume the RX data buffer is at 0x00001000 in main memory. Write 0xB000 to RX_BD_Status. Write 0x0000 to RX_BD_Length (not required because it is only done for instructional purposes). Write 0x00001000 to RX_BD_Pointer.
13. Initialize the TX buffer descriptor. Assume the TX data buffer is at 0x00002000 in main memory and contains five 8-bit characters. Write 0xBC00 to TX_BD_Status. Write 0x0005 to TX_BD_Length. Write 0x00002000 to TX_BD_Pointer.
14. Write 0xFFFF to the SCCE–Transparent to clear any previous events.
15. Write 0x0013 to the SCCM–Transparent to enable the TXE, TX, and RX interrupts.
16. Write 0x20000000 to the CIMR to allow SCC2 to generate a system interrupt. The CICR should also be initialized.

17. Write 0x00009980 to the GSMR_H to configure the transparent channel. The CDS and CTSS bits must be set to one. The TDCR, RDCR, RENC, and TENC fields must be set to zero.
18. Write 0x00000000 to the GSMR_L. Normal operation of the transmit clock is used (TCI bit is cleared). Notice that the transmitter (ENT) and receiver (ENR) have not been enabled yet.
19. Write 0x031c to the IRSIP register. When working with Timer 2 as the SIP trigger, the value should be 0x231c.
20. Write 0x0005 to the IRMODE register to enable the infra-red and to set the mode of operation to high speed.
21. Program TMR2 register when working with Timer 2 as the SIP trigger.
22. Write 0x00000030 to the GSMR_L to enable the SCC2 transmitter and receiver. This additional write ensures that the ENT and ENR bits will be enabled last.

Note: After 5 bytes have been transmitted, the TX buffer descriptor is automatically closed. Once a complete frame is received, the RX buffer descriptor is closed. Any data received after 16 bytes or a single frame causes a busy (out-of-buffers) condition since only one RX buffer descriptor is prepared.

16.9.21 The SCC2 in Transparent Mode

The SCC2 in Transparent mode allows serial data to be transmitted and received over the serial communication controller without any modification to the datastream. Transparent mode provides a clear channel on which the serial communication controller does not perform bit-level manipulation. Any protocol that uses the transparent mode must have a software layer that loads the parameters. SCC2 in Transparent mode functions as a high-speed serial-to-parallel and parallel-to-serial converter. This mode is also referred to as a totally transparent or promiscuous operation.

There are several basic applications for transparent mode. First, some data needs to be moved serially, but requires no superimposed protocol. Second, some board-level applications require a serial-to-parallel and parallel-to-serial conversion that allows communication between chips on the same board. Third, some applications require the data to be switched without interfering with the protocol encoding itself. For instance, in a multiplexer, data from a high-speed time-multiplexed serial stream is multiplexed into multiple low-speed datastreams. The objective is to switch the data path without altering the protocol encoded on that data path.

By appropriately setting the GSMR_L, the SCC2 channels can be configured to function in Transparent mode. The MPC823 receives and transmits the entire serial bitstream transparently. This mode is configured by selecting the TTX and TRX bits in the GSMR_H for the transmitter and receiver, respectively. However, both bits must be set for full-duplex transparent operation.

If just one of the TTX or TRX bits is set, the other half of the serial communication controller operates with another protocol as programmed in the MODE field of the GSMR_L. This allows loopback modes to DMA data from one memory location to another while converting the data to a specific serial format. The SCC2 in Transparent mode can work with the time-slot assigner or nonmultiplexed serial interface and support modem lines with the general-purpose I/O pins. The data can be transmitted and received with the MSB or LSB first in each octet.

The SCC2 in Transparent mode consists of separate transmit and receive sections whose operations are asynchronous with the core. Each clock can be supplied from the internal baud rate generator bank, DPLL output, or external pins.

16.9.21.1 FEATURES. The following list summarizes the main features of the SCC2 in Transparent mode:

- Flexible data buffers
- Automatic Sync detection on reception
- CRCs can be transmitted and received
- Reverse data mode
- Another protocol can be performed on the other half of the SCC2 in Transparent mode

16.9.21.2 SCC2 TRANSPARENT CHANNEL FRAME TRANSMISSION PROCESS. The transparent transmitter is designed to work with almost no intervention from the core and when the core enables the SCC2 transmitter in transparent mode, it starts transmitting idles. The serial communication controller polls the first buffer descriptor in the channel's transmit (TX) buffer descriptor table. When there is a message to transmit, the serial communication controller fetches the data from memory, loads the transmit FIFO, and waits for transmitter synchronization before transmitting the message.

Transmitter synchronization can be achieved with the $\overline{\text{CTS}}$ pin or by waiting for the receiver to achieve synchronization, depending on how the TXSY bit is set in the GSMR_H. Once transmitter synchronization is achieved, transmission begins.

When buffer descriptor data has been completely transmitted, the L bit is checked and if it is set, the serial communication controller writes the message status bits into the buffer descriptor and clears the R bit. It then starts transmitting idles until the next buffer descriptor is ready and if it is ready some idles are still transmitted. The transmitter only begins transmission again after it achieves synchronization. When the end of the current buffer descriptor has been reached and the L bit is cleared, only the R bit is cleared and the transmitter moves immediately to the next buffer to begin transmission with no gap on the serial line between buffers. Failure to provide the next buffer in time results in a transmit underrun, thus causing the TXE bit in the SCCE–Transparent register to be set.

In both cases, an interrupt is issued according to the I bit in the buffer descriptor. By appropriately setting the I bit in each buffer descriptor, interrupts are generated after each buffer or group of buffers is transmitted. The serial communication controller then proceeds to the next buffer descriptor in the table and any whole number of bytes can be transmitted. If the REVD bit in the GSMR_H is set, each data byte is reversed in its bit order before being transmitted and the most-significant bit of each octet is transmitted first.

You can decrease the latency of the transmitter by decreasing the transmit FIFO size. This option is enabled by the TFL bit in the GSMR_H and causes transmitter underruns at higher transmission speeds. An optional CRC can be appended to each transparent frame if it is enabled in the TX buffer descriptor. The CRC pattern is chosen in the TCRC field of the GSMR_H.

16.9.21.3 SCC2 TRANSPARENT CHANNEL FRAME RECEPTION PROCESS. When the core enables the SCC2 receiver in transparent mode, it waits to achieve synchronization before data is received. The receiver can be synchronized to the data by a synchronization pulse or Sync pattern.

After a buffer is filled, the serial communication controller clears the E bit in the buffer descriptor and generates an interrupt if the I bit in the buffer descriptor is set. It then moves to the next RX buffer descriptor in the table and begins moving data to its associated buffer. If the next buffer is not available as needed, the BSY bit in the SCCE–Transparent register signifies a busy signal that can generate a maskable interrupt. The receiver reverts to hunt mode when the **ENTER HUNT MODE** command or an error is received. If the REVD bit in the GSMR_H is set, each data byte is reversed in its bit order before it is written to memory.

You can decrease the latency of the receiver by decreasing the receive FIFO width. This option is enabled by the RFW bit in the GSMR_H and causes receiver overruns at higher transmission speeds. The receiver always checks the CRC of the received frame, according to the TCRC field in the GSMR_H. If a CRC is not required, the resulting errors can be ignored.

16.9.21.4 ACHIEVING SYNCHRONIZATION IN TRANSPARENT MODE. Once the SCC2 transmitter is enabled for transparent operation in the GSMR_H, the TX buffer descriptor is prepared and the transmit FIFO is preloaded by the SDMA channel, another process must occur before data can be transmitted. It is called transmit synchronization. Similarly, once the SCC2 receiver is enabled for transparent operation in the GSMR_H and the RX buffer descriptor is made empty for the serial communication controller, another process must occur before data can be received and it is called receive synchronization. You can have bit-level control of the synchronization process when receiving and transmitting by using either an inline synchronization pattern or external synchronization signals.

16.9.21.4.1 Inline Synchronization Pattern. The transparent channel can be programmed to transmit and receive a synchronization pattern if the SYNL field in the GSMR_H are not zero. This pattern is defined in the data synchronization register and the length of the Sync pattern is defined in the SYNL field. If the SYNL field is 00, then the DSR is not used and an external Sync signal is used instead. See **Section 16.9.4 Data Synchronization Register** and **Section 16.9.2 The General SCC2 Mode Registers** for more information.

DSR–SCC2 TRANSPARENT (SYNL = 01)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	4-BIT SYNC				X											
RESET	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1	0
R/W	R/W				R/W											
ADDR	(IMMR & 0xFFFF0000) + 0xA2E															

NOTE: X = "Don't Care".

DSR–SCC2 TRANSPARENT (SYNL = 10)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	8-BIT SYNC								X							
RESET	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1	0
R/W	R/W								R/W							
ADDR	(IMMR & 0xFFFF0000) + 0xA2E															

NOTE: X = "Don't Care".

DSR–SCC2 TRANSPARENT (SYNL = 11)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	16-BIT SYNC															
RESET	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1	0
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0xA2E															

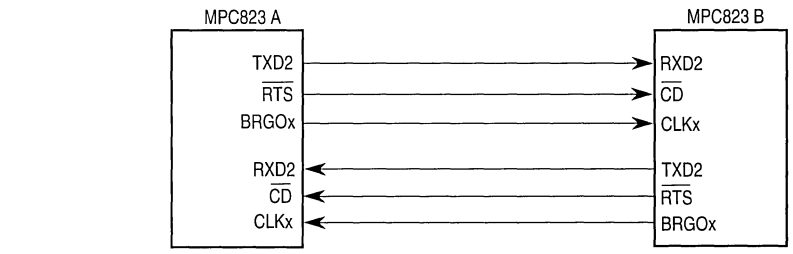
The receiver synchronizes on the synchronization pattern that is located in the DSR. For instance, if 4-BIT SYNC is selected, reception begins as soon as these four bits are received, beginning with the first bit following the 4-BIT SYNC field. The transmitter synchronizes on the receiver pattern if the RSYN bit of the GSMR_H is set. This effectively links the transmitter synchronization to the receiver synchronization.

16.9.21.4.2 External Synchronization Signals. If the SYNL field of the GSMR_H is programmed to 00, an external signal is used to begin the sequence. The $\overline{CTS}$ pin is used for the transmitter and the $\overline{CD}$ pin is used for the receiver and these pins share two options—pulse and sampling.

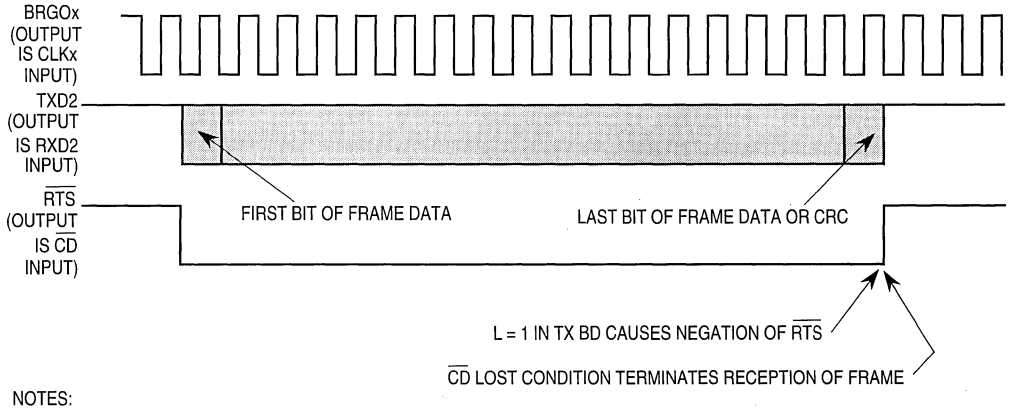
The pulse option determines whether the $\overline{CD}$ or $\overline{CTS}$ pins only need to be asserted once to begin reception/transmission or whether they must be asserted and stay that way for the duration of the transparent frame. In the GSMR_H, the CDP and CTSP bits control the pulse options. If you expect a continuous stream of data without interruption, then you should use the pulse operation. However, if you are trying to identify frames of transparent data, then you should use the envelope mode of these pins.

The sampling option determines the delay between $\overline{CD}$ and $\overline{CTS}$ being asserted and the resulting action by the serial communication controller. You can assume that these pins are asynchronous to the data and internally synchronized by the serial communication controller or you can assume that they are synchronous to the data for faster operation. The CDS and CTCC bits of the GSMR_H control the sampling option. It is also an option to link the transmitter synchronization to the receiver synchronization. Diagrams for the pulse/envelope and sampling options are illustrated in **Section 16.9.10 Controlling SCC2 Timing**.

16.9.21.4.3 Transparent Synchronization Example. Figure 16-102 illustrates an example of synchronization using the external signals.



NOTES:
1. Each MPC823 generates its own transmit clocks. If the transmit and receive clocks are the same, one MPC823 can generate transmit and receive clocks for the other MPC823. For example, CLKx on MPC823 B could be used to clock the transmitter and receiver.



NOTES:
1. CTS should be configured as always asserted in the port C parallel I/O or connected to ground externally.
2. The required GSMR_x configurations are DIAG = 00, CTSS = 1, CTSP is a "don't care", CDS = 1, CDP = 0, TTX = 1, and TRX = 1. REVD and TCRC are application-dependent.
3. The transparent frame contains a CRC if the TC bit is set in the TX buffer descriptor.

Figure 16-102. Sending Transparent Frames Between MPC823s

MPC823 A and B in Figure 16-102 exchange transparent frames and synchronize each other using the RTS and CD pins. However, the CTS pin is not required since transmission begins at any time. Thus, the RTS pin is directly connected to the other CD pin. The RSYN bit in the GSMR_H is not set and transmission and reception from each MPC823 is independent.

16.9.21.5 SCC2 TRANSPARENT PARAMETER RAM MEMORY MAP. In totally Transparent mode, the serial communication controller overlays the structure used in Table 16-24 with the transparent parameters described in Table 16-29.

Table 16-29. SCC2 Transparent Parameter RAM Memory Map

ADDRESS	NAME	WIDTH	DESCRIPTION
SCC2 Base + 30	**CRC_P**	Long	CRC Preset for Totally Transparent Mode
SCC2 Base + 34	**CRC_C**	Long	CRC Constant for Totally Transparent Receiver

NOTE: You are only responsible for initializing the items in bold.
SCC2 Base = (IMMR & 0xFFFF0000) + 0x3D00.

- CRC_P—For the 16-bit CRC-CCITT, CRC_P should be initialized with 0x0000FFFF. For the 32-bit CRC-CCITT, CRC_P should be initialized with 0xFFFFFFFF and for the CRC-16, CRC_P should be initialized with ones (0x0000FFFF) or zeros (0x00000000).
- CRC_C—For the 16-bit CRC-CCITT, CRC_C should be initialized with 0x0000F0B8. For the 32-bit CRC-CCITT, CRC_C should be initialized with 0xDEBB20E3 and for the CRC-16 which is normally used with BISYNC, CRC_C should be initialized with 0x00000000.

Note: CRC-C overlaps with the CRC constant for the HDLC-based protocols. However, this overlap is not detrimental since the CRC constant is only used for the receiver, so only one entry is required. You can choose an HDLC transmitter with a transparent receiver or a transparent transmitter with an HDLC receiver.

16.9.21.6 SCC2 TRANSPARENT COMMANDS. You can program the CPM command register (CPCR) with the following commands to transmit data. See **Section 16.2.7.1 CPM Command Register** for additional information.

- **STOP TRANSMIT**—After the hardware or software is reset and the channel is enabled in the SCC2 mode register, the channel is in transmit enable mode and starts polling the first buffer descriptor in the table every 64 clocks or immediately if the TOD bit of the TODR is set. This command disables the transmission of frames on the transmit channel and if the transparent controller receives it during frame transmission, buffer transmission is aborted after a maximum of 64 additional bits and the transmit FIFO is flushed. The TBPTR is not advanced, no new buffer descriptor is accessed and no new buffers are transmitted for this channel. The transmitter will send idles.

- **GRACEFUL STOP TRANSMIT**—This command is used to stop transmission smoothly, rather than abruptly, in much the same way that the regular **STOP TRANSMIT** command stops. It stops transmission after the current frame finishes or immediately if there is no frame being transmitted. A transparent frame is not complete until a buffer descriptor with the L bit set has its associated buffer completely transmitted. The GRA bit in the SCCE–Transparent register is set once transmission stops and then the transmit parameters and their buffer descriptors can be modified. The TBPTR points to the next TX buffer descriptor in the table. Transmission begins once the R bit of the next buffer descriptor is set and the **RESTART TRANSMIT** command is issued.
- **RESTART TRANSMIT**—This command reenables the transmission of characters on the transmit channel. The transparent controller expects it after a **STOP TRANSMIT** command is issued, after a **GRACEFUL STOP TRANSMIT** command is issued, or after a transmitter error occurs. The transparent controller resumes transmission from the current TBPTR in the channel TX buffer descriptor table.
- **INIT TX PARAMETERS**—This command initializes all transmit parameters in the serial channel parameter RAM to their reset state. It should only be issued when the transmitter is disabled. The **INIT TX AND RX PARAMS** command can also be used to reset the transmit and receive parameters.

You can program the CPM command register with the following commands to receive data. See **Section 16.2.7.1 CPM Command Register** for additional information.

- **ENTER HUNT MODE**—After the hardware or software is reset and the channel is enabled in the GSMR_H, the channel is in receive enable mode and uses the first buffer descriptor in the table. This command is used to force the transparent receiver to stop receiving the current frame and enter hunt mode where the transparent controller is waiting for the synchronization sequence. After receiving the command, the current receive buffer is closed. Further data reception uses the next buffer descriptor.
- **CLOSE RX BD**—This command is used to force the serial communication controller to close the RX buffer descriptor if it is currently being used and to use the next buffer descriptor for any subsequently received data. If the serial communication controller is not in the process of receiving data, no action is taken by this command.
- **INIT RX PARAMETERS**—This command initializes all the receive parameters in this serial channel parameter RAM to their reset state and should only be issued when the receiver is disabled. The **INIT TX AND RX PARAMS** command can also be used to reset the receive and transmit parameters.

16.9.21.7 SCC2 TRANSPARENT CONTROLLER ERRORS. The serial communication controller reports message reception and transmission errors using the channel buffer descriptors, the error counters, and the SCCE–Transparent register. The following transmission errors can be detected by the SCC2 Transparent controller.

- Transmitter Underrun—When this error occurs, the channel stops transmitting the buffer, closes it, sets the UN bit of the buffer descriptor, and generates the TXE interrupt if it is enabled. The channel resumes transmission after the **RESTART TRANSMIT** command is received. Underrun occurs after a transmit frame for which the L bit of the TX buffer descriptor was not set. In this case, only the TXE bit is set. Underrun cannot occur between transparent frames.
- $\overline{\text{CTS}}$ Lost During Message Transmission—When this error occurs, the channel stops transmitting the buffer, closes it, sets the CT bit of the buffer descriptor, and generates the TXE interrupt if it is enabled. The channel resumes transmission after the **RESTART TRANSMIT** command is received.

The following reception errors can be detected by the SCC2 Transparent controller.

- Overrun—The serial communication controller maintains an internal FIFO for receiving data. The communication processor module starts programming the SDMA channel if the data buffer is in external memory and updating the CRC when 8 or 32 bits are received in the FIFO. If a FIFO overrun occurs, the serial communication controller writes the received data byte to the internal FIFO over the previously received byte. The previous character and its status bits are lost. Afterwards, the channel closes the buffer, sets the OV bit of the buffer descriptor, and generates the RX interrupt if it is enabled. The receiver immediately enters hunt mode.
- $\overline{\text{CD}}$ Lost During Message Reception—When this error occurs, the channel stops receiving messages, closes the buffer, sets the CD bit of the buffer descriptor, and generates the RX interrupt if it is enabled. This error has the highest priority, the rest of the message is lost, and no other errors are checked in the message.

16.9.21.8 SCC2 TRANSPARENT MODE REGISTER. Since all transparent mode selections are in the GSMR_H, the PSMR is not used by the transparent controller. If transparent mode is only selected for the transmitter/receiver, then the transmitter/receiver can be programmed to support another protocol. In such a case, you can use the PSMR for the other protocol.

16.9.21.9 SCC2 TRANSPARENT RECEIVE BUFFER DESCRIPTOR. The communication processor module reports information about the received data for each buffer using a channel's receive (RX) buffer descriptor, closes the current buffer, generates a maskable interrupt, and starts receiving data into the next buffer after one of the following events occurs:

- An error is detected.
- A full receive buffer is detected.
- The **ENTER HUNT MODE** command is issued.
- The **CLOSE RX BD** command is issued.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	**E**	RES	**W**	**I**	L	F	**CM**	RES	DE	RESERVED		NO	RES	CR	OV	CD
OFFSET + 2	**DATA LENGTH**															
OFFSET + 4	**RX DATA BUFFER POINTER**															
OFFSET + 6																

NOTE: You are only responsible for initializing the items in bold.

E—Empty

0 = The data buffer associated with this RX buffer descriptor has been filled with data or has stopped receiving data because an error occurred. The core is free to examine or write to any fields of this RX buffer descriptor. The communication processor module does not use this buffer descriptor when the E bit is zero.

1 = The data buffer associated with this buffer descriptor is empty or currently receiving data. This RX buffer descriptor and its associated receive buffer are owned by the communication processor module. Once the E bit is set, the core should not write any fields of this RX buffer descriptor.

Bits 1, 7, 9–10, and 12—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the RX buffer descriptor table.
1 = This is the last buffer descriptor in the RX buffer descriptor table. After this buffer is used, the communication processor module receives incoming data into the first buffer descriptor that RBASE points to in the table. The number of RX buffer descriptors in this table is programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is used.
1 = When this buffer is closed by the transparent controller, the RX bit in the transparent event register is set. The RX bit can cause an interrupt if it is enabled.

L—Last in Frame

This bit is set by the transparent controller when this buffer is the last in a frame. If CD in envelope mode is negated or an error has been received, one or more of the OV, CD, and DE bits are set. The transparent controller writes the number of frame octets to the DATA LENGTH field.

0 = This buffer is not the last one in a frame.
1 = This buffer is the last one in a frame.

F—First in Frame

The transparent controller sets this bit when this buffer is the first in the frame:

0 = The buffer is not the first in a frame.
1 = The buffer is the first in a frame.

CM—Continuous Mode

0 = Normal operation.
1 = The E bit is not cleared by the communication processor module after this buffer descriptor is closed, thus allowing the associated data buffer to be automatically overwritten next time the communication processor module accesses this buffer descriptor. However, the E bit is cleared if an error occurs during reception, regardless of how the CM bit is set.

DE—DPLL Error

This bit is set by the transparent controller when a DPLL error occurs while this buffer is being received. In decoding modes, where a transition occurs on every bit, the DPLL error is set when a missing transition occurs.

NO—RX Non-Octet

This bit indicates that a frame containing a number of bits not exactly divisible by 8 is received.

CR—CRC Error indication bits

This bit indicates that this frame contains a CRC error. The received CRC bytes are always written to the receive buffer.

OV—Overrun

This bit indicates that a receiver overrun has occurred during buffer reception.

CD—Carrier Detect Lost

This bit indicates when the $\overline{CD}$ pin is negated during buffer reception.

DATA LENGTH

This field represents the number of octets that the communication processor module writes into this buffer descriptor data buffer. The communication processor module only writes it once as the buffer is closed. The actual amount of memory allocated for this buffer should be greater than or equal to MRBLR.

RX DATA BUFFER POINTER

This field always point to the first location of the associated data buffer and must be divisible by four, unless the RFW bit in the GSMR_H is set to 8 bits wide, even or odd. The buffer can reside in internal or external memory.

16.9.21.10 SCC2 TRANSPARENT TRANSMIT BUFFER DESCRIPTOR. Data is sent to the communication processor module for transmission on an SCC2 channel by arranging it in buffers referenced by the channel's transmit (TX) buffer descriptor table. Using the buffer descriptors, the communication processor module confirms transmission or indicates error conditions so that the processor knows the buffers have been serviced. You must prepare the status and control bits before transmission, but they are set by the communication processor module after the buffer has been transmitted.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	R	RES	W	I	L	TC	CM	RESERVED							UN	CT
OFFSET + 2	**DATA LENGTH**															
OFFSET + 4	**TX DATA BUFFER POINTER**															
OFFSET + 6																

NOTE: You are only responsible for initializing the items in bold.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

R—Ready

0 = The data buffer associated with this buffer descriptor is not ready for transmission. You are free to manipulate this buffer descriptor or its associated data buffer. The communication processor module clears this bit after the buffer is transmitted or after an error condition is encountered.

1 = The data buffer, which you have prepared for transmission, is not transmitted yet or is currently being transmitted. You cannot write any fields of this buffer descriptor once this bit is set.

Bits 1 and 7–13—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the TX buffer descriptor table.

1 = This is the last buffer descriptor in the TX buffer descriptor table. After this buffer is used, the communication processor module receives incoming data into the first buffer descriptor that TBASE points to in the table. The number of TX buffer descriptors in this table is programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is serviced.

1 = When this buffer is serviced by the communication processor module, the TX or TXE bit is set in the SCCE–Transparent register. These bits can cause interrupts if they are enabled.

L—Last in Message

0 = The last byte in the buffer is not the last byte in the transmitted transparent frame. Data from the next transmit buffer is transmitted immediately after the last byte of this buffer.

1 = The last byte in the buffer is the last byte in the transmitted transparent frame. After this buffer is transmitted, the transmitter requires synchronization before the next buffer is transmitted.

TC—Transmit CRC

0 = No CRC sequence is transmitted after this buffer.

1 = A frame check sequence that is defined by the TCRC field in the GSMR_H is transmitted after the last byte of this buffer.

CM—Continuous Mode

0 = Normal operation.

1 = The R bit is not cleared by the communication processor module after this buffer descriptor is closed, thus allowing the associated data buffer to be automatically retransmitted next time the communication processor module accesses this buffer descriptor. However, the R bit is cleared if an error occurs during transmission, regardless of how the CM bit is set.

UN—Underrun

This bit indicates that the serial communication controller has encountered a transmitter underrun condition while transmitting the associated data buffer.

CT—$\overline{\text{CTS}}$ Lost

This bit indicates the $\overline{\text{CTS}}$ signal has been lost during frame transmission.

DATA LENGTH

This field represents the number of bytes that the communication processor module should transmit from this buffer descriptor data buffer. It should be greater than zero and can be even or odd. This value is never modified by the communication processor module.

TX DATA BUFFER POINTER

This field always points to the first byte of the associated data buffer. It can be even or odd, and can reside in internal or external memory.

16.9.21.11 SCC2 TRANSPARENT EVENT REGISTER. When the SCC2 is in transparent mode, the 16-bit, memory-mapped SCC2 event register is referred to as the SCC2 transparent event register (SCCE–Transparent). Since each protocol has specific requirements, the SCCE bits are different for each implementation. This register is used to report events recognized by the transparent channel and to generate interrupts. When an event is recognized, the transparent controller sets the corresponding bit in this register. Interrupts generated by this register can be masked in the SCCM–Transparent register.

A bit is reset by writing a 1 (writing a zero has no effect) and more than one bit can be reset at a time. All unmasked bits must be reset before the communication processor module negates the internal interrupt request signal. This register is cleared at reset and can be read at any time.

SCCE–TRANSPARENT

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED			GLR	GLT	DCC	RES		GRA	RES		TXE	RCH	BSY	TX	RX
RESET	0			0	0	0	0		0	0		0	0	0	0	0
R/W	R/W			R/W	R/W	R/W	R/W		R/W	R/W		R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA30															

Bits 0–2, 6–7, and 9–10—Reserved

These bits are reserved and should be set to 0.

GLR—Glitch on RX

This bit indicates that the the serial communication controller has found a glitch on the receive clock.

GLT—Glitch on TX

This bit indicates that the the serial communication controller has found a glitch on the transmit clock.

DCC—DPLL CS Changed

This bit indicates when the carrier sense status that is generated by the DPLL changes state. The real-time status can be found in the SCCS–Transparent register. This is not the $\overline{CD}$ pin status that is mentioned elsewhere and it is only valid when the DPLL is used.

GRA—Graceful Stop Complete

This bit indicates when a graceful stop initiated by the **GRACEFUL STOP TRANSMIT** command has completed. This bit is set as soon as the transmitter finishes any frame that was in progress when the command was issued. It is set immediately if no frame was in progress when the command was issued.

TXE—TX Error

This bit indicates that an error has occurred on the transmitter channel. The I bit in the transmit buffer descriptor must be set in order to get an update of this bit.

RCH—Receive Character

This bit indicates that a byte or word has been received and written to the buffer, depending on how the RFW bit is set in the GSMR_H.

BSY—Busy Condition

This bit indicates that a byte or word has been received and discarded due to a lack of buffers. The receiver resumes reception after it gets an **ENTER HUNT MODE** command.

TX—TX Buffer

This bit indicates that a buffer has been transmitted. This bit is set no sooner than when the last bit of the last byte of the buffer begins its transmission, assuming the L bit of the TX buffer descriptor is set. If it is not set, TX is set when the last byte of data is written to the transmit FIFO. The I bit in the transmit buffer descriptor must be set in order to get an update of this bit.

RX—RX Buffer

This bit indicates that a complete buffer has been received on the SCC2 channel. This bit is set no sooner than two serial clocks after the last bit of the last byte in which the buffer is received on the RXD2 pin. The I bit in the receive buffer descriptor must be set in order to get an update of this bit.

16.9.21.12 SCC2 TRANSPARENT MASK REGISTER. When the SCC2 is in transparent mode, the 16-bit read/write SCC2 mask register is referred to as the SCC2 transparent mask (SCCM–Transparent) register. Since each protocol has specific requirements, the SCCM bits are different for each implementation. It has the same bit format as the SCCE–Transparent register. If a bit in this register is 1, the corresponding interrupt in the this register is enabled. If the bit is zero, the corresponding interrupt in the this register is masked.

SCCM–TRANSPARENT

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED			GLR	GLT	DCC	RES		GRA	RES		TXE	RCH	BSY	TX	RX
RESET	0			0	0	0	0		0	0		0	0	0	0	0
R/W	R/W			R/W	R/W	R/W	R/W		R/W	R/W		R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA34															

16.9.21.13 SCC2 TRANSPARENT STATUS REGISTER. When the SCC2 is in transparent mode, the 8-bit read-only SCC2 status register is referred to as the SCC2 transparent status (SCCS–Transparent) register. Since each protocol has specific requirements, the SCCM bits are different for each implementation. This register allows you to monitor real-time status conditions on the RXD2 line. The real-time status of the $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ pins are part of the port C parallel I/O. Since each protocol has specific requirements, the SCCS bits are different for each implementation.

SCCS–TRANSPARENT

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED						CS	RESERVED
RESET	0						0	0
R/W	R						R	R
ADDR	(IMMR & 0xFFFF0000) + 0xA37							

Bits 0–5 and 7—Reserved

These bits are reserved and should be set to 0.

CS—Carrier Sense (DPLL)

This bit shows the real-time internal $\overline{\text{CS}}$ signal, as determined by the DPLL.

0 = The DPLL does not sense a carrier.
1 = The DPLL senses a carrier.

16.9.21.14 SCC2 TRANSPARENT PROGRAMMING EXAMPLE. The following is an example initialization sequence for SCC2 in transparent mode. The transmitter and receiver are both enabled, but operate independently of each other. They implement the connection illustrated on MPC823(B) in Figure 16-102. The transmit and receive clocks are externally provided to MPC823(B) using the CLK3 pin. SCC2 is used. The transparent controller is configured with the $\overline{\text{RTS2}}$ and $\overline{\text{CD2}}$ pins active and $\overline{\text{CTS2}}$ is grounded internally by the configuration in port C. A 16-bit CRC-CCITT is sent with each transparent frame. The FIFOs are configured for fast operation.

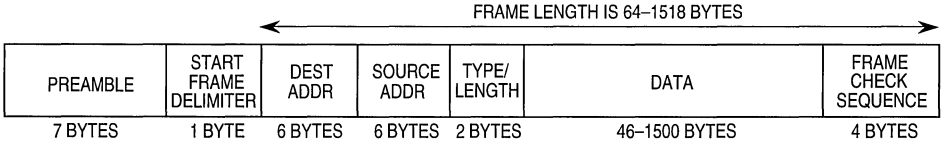
1. Configure the port A pins to enable the TXD2 and RXD2 pins. Write PAPAR bits 13 and 12 with ones and then PADIR and PAODR bits 13 and 12 with zeros.
2. Configure the port C pins to enable $\overline{\text{RTS2}}$, $\overline{\text{CTS2}}$, and $\overline{\text{CD2}}$. Write PCPAR bit 14 with one and bits 8 and 9 with zero, PCDIR bits 14, 9, and 8 with zero, and PCSO bits 8 and 9 with one.
3. Configure port A to enable the CLK3 pin. Write PAPAR bit 5 with a one and PADIR bit 5 with a zero.
4. Connect the CLK3 pin to SCC2 using the serial interface. Write the R2CS and T2CS bits of the SICR to 110.
5. Connect the SCC2 to the NMSI and clear the SC2 bit of the SICR.
6. Write the SDCR with the appropriate arbitration ID.
7. Write RBASE and TBASE in the SCC2 parameter RAM to point to the RX buffer descriptor and TX buffer descriptor in the dual-port RAM. Assuming one RX buffer descriptor at the beginning of dual-port RAM and one TX buffer descriptor following that RX buffer descriptor, write RBASE with 0x2000 and TBASE with 0x2008.
8. Program the CPCR to execute the **INIT RX AND TX PARAMS** command for the serial communication controller. To execute this command for SCC2, write 0x0041 to the CPCR.
9. Write RFCR and TFCR with 0x18 for normal operation.
10. Write MRBLR with the maximum number of bytes per receive buffer and assume 16 bytes, so MRBLR = 0x0010.
11. Write CRC_P with 0x0000FFFF to comply with the 16-bit CRC-CCITT.
12. Write CRC_C with 0x0000F0B8 to comply with the 16-bit CRC-CCITT.
13. Initialize the RX buffer descriptor. Assume the RX data buffer is at 0x00001000 in main memory. Write 0xB000 to RX_BD_Status, 0x0000 to RX_BD_Length (optional), and 0x00001000 to RX_BD_Pointer.
14. Initialize the TX buffer descriptor. Assume the TX data buffer is at 0x00002000 in main memory and contains five 8-bit characters. Write 0xBC00 to TX_BD_Status, 0x0005 to TX_BD_Length, and 0x00002000 to TX_BD_Pointer.
15. Write 0xFFFF to the SCCE–Transparent to clear any previous events.
16. Write 0x0013 to the SCCM–Transparent to enable the TXE, TX, and RX interrupts.
17. Write 0x20000000 to the CIMR so the SCC2 can generate a system interrupt. The CICR should also be initialized.

18. Write 0x00001980 to the GSMR_H to configure the transparent channel.
19. Write 0x00000000 to the GSMR_L to configure the $\overline{CTS}$ and $\overline{CD}$ pins to automatically control transmission and reception (DIAG field). Normal operation of the transmit clock is used. Notice that the transmitter (ENT) and receiver (ENR) are not enabled yet.
20. Write 0x00000030 to the GSMR_L to enable the SCC2 transmitter and receiver. This additional write ensures that the ENT and ENR bits are enabled last.

Note: After 5 bytes are transmitted, the TX buffer descriptor is automatically closed. The receive buffer is automatically closed after 16 bytes are received. Any data received after 16 bytes causes a busy (out-of-buffers) condition since only one RX buffer descriptor is prepared.

16.9.22 The SCC2 in Ethernet Mode

The Ethernet IEEE 802.3 protocol is a widely used LAN-based on the carrier-sense multiple access/collision detect (CSMA/CD) approach. Ethernet and IEEE 802.3 protocols are very similar and can coexist on the same LAN. They are referred to synonymously as Ethernet in this manual, unless specifically noted. Ethernet and IEEE 802.3 frames are based on the frame structure illustrated in Figure 16-103.



NOTE: The LSB of each octet is transmitted first.

Figure 16-103. Ethernet Frame Format

The frame begins with a 7-byte preamble of alternating ones and zeros. Since the frame is Manchester encoded, the preamble gives receiving stations a known pattern on which to lock. The start frame delimiter signifies the beginning of the frame and follows the preamble. The 48-bit destination address is next, followed by the 48-bit source address. Original versions of the IEEE 802.3 specification allowed 16-bit addressing. However, this addressing has never been widely used in the industry.

The next field is the type field in Ethernet and the length field in IEEE 802.3. The type field signifies the protocol used in the rest of the frame and the length field specifies the length of the data portion of the frame. For Ethernet and IEEE 802.3 frames to coexist on the same LAN, the length field of the frame must always be unique from any type fields used in Ethernet. This has limited the length of the data portion of the frame to 1,500 bytes and the total frame length to 1,518 bytes. The last 4 bytes of the frame are the frame check sequence (FCS), which is the standard 32-bit CCITT-CRC polynomial used in many other protocols.

When a station needs to transmit, it checks for activity on the LAN and when the LAN becomes silent for a specified period, the station starts transmitting. At that time, the station continually checks for collision on the LAN and if one is found, the station forces a jam of all ones on its frame and stops transmitting. Collisions usually occur close to the beginning of a frame. The station waits a random period of time, called a backoff, before trying to transmit again. Once the backoff is complete, the station waits for silence on the LAN and then begins retransmission on the LAN, which is called a retry. If the frame is not successfully transmitted within 15 retries, an error occurs. The 10Mbps Ethernet provides 0.8µs per byte. The preamble plus start frame delimiter is transmitted in 6.4µs. The minimum interframe gap is 9.6µs and the slot time is 52µs. Therefore, you must operate the MPC823 at a minimum frequency of 20MHz to implement Ethernet.

16.9.22.1 FEATURES. The following list summarizes the main features of the SCC2 in Ethernet mode:

- Performs MAC layer functions of Ethernet and IEEE 802.3
- Full-duplex operation support
- Performs framing functions
- Full collision support
- Maximum 10Mbps (10BASE-T) bit rate
- Back-to-back frame reception
- Detection of receive frames that are too long
- Multibuffer data structure
- Supports 48-bit addresses in three modes
- Up to eight parallel I/O pins can be sampled and appended to any frame
- Heartbeat indication
- Transmitter network management and diagnostics
- Receiver network management and diagnostics
- Error counters
- Internal and external loopback mode

16.9.22.2 ETHERNET ON THE MPC823. When the MODE field in the general SCC2 mode low register (GSMR_L) selects the SCC2 to be in Ethernet mode, the serial communication controller performs the full set of IEEE 802.3/Ethernet CSMA/CD media access control and channel interface functions. The SCC2 in Ethernet mode is also referred to as the SCC2 Ethernet controller in this manual.

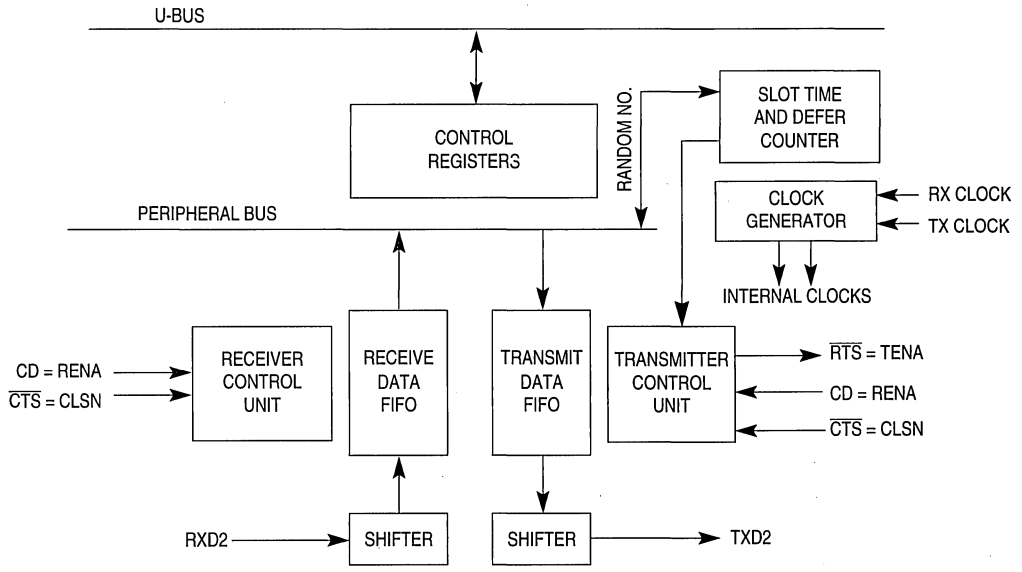


Figure 16-104. Ethernet Block Diagram

The SCC2 in Ethernet mode requires an external serial interface adaptor (SIA) and transceiver function to complete the interface to the media. This function is implemented in the Motorola MC68160 enhanced Ethernet serial transceiver (EEST).

The MPC823+EEST solution provides a direct connection to the attachment unit interface (AUI) or twisted-pair (10BASE-T). The EEST provides a glueless interface to the MPC823, Manchester encoding and decoding, automatic selection of 10BASE-T versus AUI ports, 10BASE-T polarity detection and correction, LED drivers, and a low-power mode. For more information, refer to the MC68160 device description.

The SCC2 Ethernet controller also provides a number of features. Although the MPC823 contains DPLLs that allow Manchester encoding and decoding, these DPLLs were not designed for Ethernet rates. Therefore, the SCC2 Ethernet controller bypasses the on-chip DPLLs and uses the external system interface adaptor on the EEST instead.

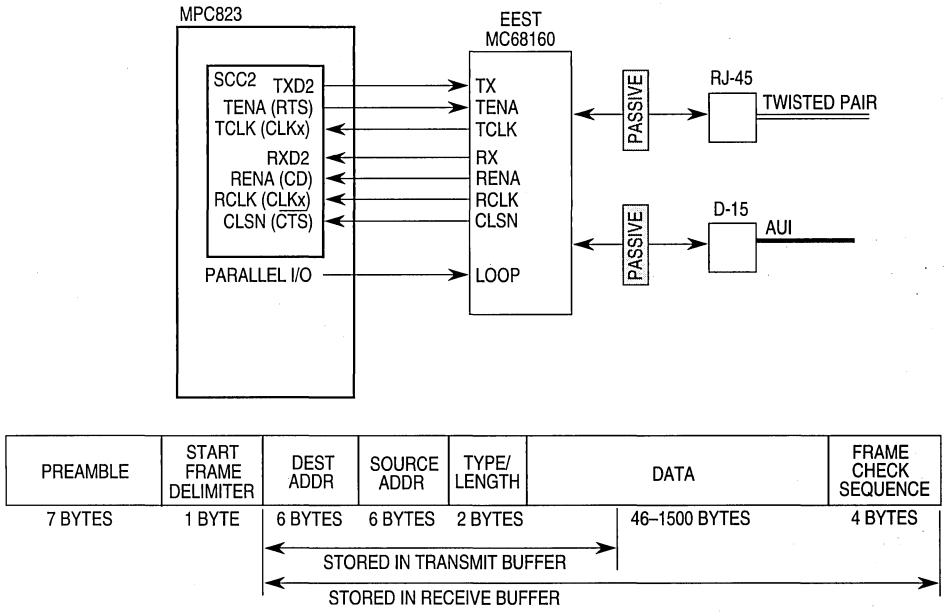
16.9.22.3 UNDERSTANDING ETHERNET ON THE MPC823. You are encouraged to learn about the basic functionality of the SCC2 and the overall architecture of the communication processor module before delving into the functionality of the SCC2 in Ethernet mode. It is important that first-time users of the MPC823 who plan to use Ethernet read the following sections of this manual first.

- **Section 16.2 The RISC Microcontroller**, **Section 16.2 The RISC Microcontroller**, and **Section 16.2.7.3 Dual-Port RAM** explain how the RISC microcontroller issues special commands to the Ethernet channel. The dual-port RAM loads Ethernet parameters and initializes buffer descriptors for the Ethernet channel to use.
- **Section 16.5 The SDMA Channels** explains how SDMA channels are used to transfer data to or from the Ethernet channel and system memory.
- **Section 16.7.8 Nonmultiplexed Serial Interface Configuration** explains how clocks are routed to the SCC through the bank of clocks.
- **Section 16.9.22 The SCC2 in Ethernet Mode** explains how to program the SCC2 in Ethernet mode.
- **Section 16.14 The Parallel I/O Ports** explains how to configure the preferred Ethernet pin functions to be active.
- **Section 16.15 The CPM Interrupt Controller** defines the interrupt priority of the serial communication controller and how interrupts are generated to the core.
- **Section 12.3 Interrupt Configuration** contains an overview of the MPC823 interrupt structure and explains how to set up interrupt control.

16.9.22.4 CONNECTING THE MPC823 TO THE EEST. The interface to the external EEST chip consists of the following Ethernet pins:

- Receive clock (RCLK)—The CLK1, CLK2, CLK3, or CLK4 pin that is routed through the bank of clocks on the MPC823.
- Transmit clock (TCLK)—The CLK1, CLK2, CLK3, or CLK4 pin that is routed through the bank of clocks on the MPC823. RCLK and TCLK should not be connected to the same CLKx pin since the EEST provides a separate receive and transmit clock signal.
- Transmit data (TXD2)—The MPC823 TXD2 pin.
- Receive data (RXD2)—The MPC823 RXD2 pin.

Figure 16-105 illustrates the basic components and pins required to make the Ethernet connection between the MPC823 and EEST.



NOTE: The MPC823 automatically pads the short transmit frames.

Figure 16-105. Connecting the MPC823 to Ethernet

The following pins function differently when the SCC2 is in Ethernet mode than when it is in other protocols:

- Transmit Enable (TENA)—The $\overline{\text{RTS}}$ pin changes to TENA when the SCC2 is in Ethernet mode. The polarity of TENA is active high, whereas the polarity of $\overline{\text{RTS}}$ is active low.
- Receive Enable (RENA)—The $\overline{\text{CD}}$ pin changes to RENA when the SCC2 is in Ethernet mode. The polarity of RENA is active high, whereas the polarity of $\overline{\text{CD}}$ is active low.
- Collision (CLSN)—The $\overline{\text{CTS}}$ pin changes to CLSN when the SCC2 is in Ethernet mode. The polarity of CLSN is active high, whereas the polarity of $\overline{\text{CTS}}$ is active low.

Note: The carrier sense signal is referenced in Ethernet descriptions because it indicates when the LAN is being used. Carrier sense is defined as RENA OR'ed with CLSN.

The EEST has similar names for its connection to the seven MPC823 pins mentioned above and contains a loopback pin so the MPC823 can perform external loopback testing. This can be controlled by any available parallel I/O pin on the MPC823. The passive components that are needed to connect to AUI or twisted-pair media are external to the EEST. For more information on the EEST connection circuits, refer to the MC68160 device description.

Using the SDMA channels, the MPC823 stores every byte that is received after the start frame delimiter in system memory. When transmitting, you provide the destination address, source address, type/length field, and the transmit data. The MPC823 automatically pads frames with less than 46 bytes in the data field to meet the minimum frame requirements. In addition, the MPC823 appends the FCS to the frame.

16.9.22.5 SCC2 ETHERNET CHANNEL FRAME TRANSMISSION PROCESS. The Ethernet transmitter is designed to work with almost no intervention from the core. When the core enables the transmitter, the SCC2 in Ethernet mode polls the first TX buffer descriptor in the channel TX buffer descriptor table every 128 serial clocks. If you have a frame to transmit, you can set the TOD bit in the TODR to avoid having to wait for the next poll to occur. See **Section 16.9.5 Transmit-on-Demand Register** for more information.

To begin transmission, the SCC2 in Ethernet mode (also called the SCC2 Ethernet controller) fetches the data from the data buffer, asserts TENA to the EEST, and starts transmitting the preamble sequence, the start frame delimiter, and frame information. However, the SCC2 Ethernet controller defers transmission if the line is busy. Before transmitting, it waits for carrier sense to become inactive and stay that way for 6.0μs. If it does, then the SCC2 Ethernet controller starts transmitting after waiting an additional 3.6μs (9.6μs after carrier sense originally became inactive).

If a collision occurs during frame transmission, the SCC2 Ethernet controller follows the specified backoff procedure and tries to retransmit the frame until the retry limit threshold is reached. The SCC2 Ethernet controller stores the first 5 to 8 bytes of the transmit frame in internal RAM, so that they do not have to be retrieved from system memory in case of a collision. This improves bus utilization and latency when the backoff timer output requires an immediate retransmission. If a collision occurs during frame transmission, the SCC2 Ethernet controller returns to the first buffer for a retransmission. The only restriction is that the first buffer must contain at least 9 bytes.

Note: If an Ethernet frame is made up of multiple buffers, you should not reuse the first buffer descriptor until the last buffer descriptor of the frame has had its R bit cleared by the communication processor module.

When the end of the current buffer descriptor is reached and the L bit in the TX buffer descriptor is set, the FCS bytes of the Ethernet frame are appended (if the TC bit is set in the TX buffer descriptor), and TENA is negated. This notifies the EEST of the need to generate the illegal Manchester encoding that signifies the end of an Ethernet frame. After CRC transmission, the SCC2 Ethernet controller writes the frame status bits into the buffer descriptor and clears the R bit. When the end of the current buffer descriptor is reached and the L bit is not set, only the R bit is cleared.

In either mode, an interrupt can be issued, depending on how the I bit is set in the TX buffer descriptor. The SCC2 Ethernet controller then proceeds to the next TX buffer descriptor in the table. You can be interrupted after each frame, after each buffer, or after a specific buffer is transmitted. The SCC2 Ethernet controller can add pad characters to short frames. If the PAD bit is set in the TX buffer descriptor, the frame is padded up to the value of the minimum frame length register.

To rearrange the transmit queue before the communication processor module finishes transmitting all the frames, issue the **GRACEFUL STOP TRANSMIT** command. This technique can be useful for transmitting expedited data before previously linked buffers or for error situations. When the **GRACEFUL STOP TRANSMIT** command is issued, the Ethernet controller stops immediately if no transmission is in progress or it will keep transmitting until the current frame either finishes or terminates with a collision. When the Ethernet controller receives the **RESTART TRANSMIT** command, it resumes transmission. The Ethernet controller transmits bytes least-significant bit first.

16.9.22.6 SCC2 ETHERNET CHANNEL FRAME RECEPTION PROCESS. The Ethernet receiver is designed to work with almost no intervention from the core and can perform address recognition, CRC checking, short frame checking, maximum DMA transfer checking, and maximum frame length checking.

When the core enables the Ethernet receiver, it enters hunt mode as soon as the RENA signal is asserted if CLSN is negated. In hunt mode, as data is shifted into the receive shift register one bit at a time, the contents of the register are compared to the contents of the SYN1 field in the data synchronization register. This compare function becomes valid a certain number of clocks after the start of the frame (depending on the NIB bits in the PSMR–Ethernet). If the two are not equal, the next bit is shifted in and the comparison is repeated. If a double zero or double one fault is detected between bits 14 to 21 from the start of the frame, it is rejected. If a double zero fault is detected after 21 bits from the start of the frame and before detection of the start frame delimiter, the frame is also rejected. When the registers match, hunt mode is terminated and character assembly begins.

When the receiver detects the first bytes of the frame, the Ethernet controller performs address recognition functions on the frame. The receiver can receive physical (individual), group (multicast), and broadcast addresses. Ethernet receive frame data is not written to memory until the internal address recognition algorithm is complete, which improves bus utilization with frames not addressed to this station.

If a match is found, the Ethernet controller fetches the next RX buffer descriptor and, if it is empty, starts transferring the incoming frame to the RX buffer descriptor associated data buffer. If a collision is detected during the frame, the RX buffer descriptors associated with this frame are reused. Thus, there will be no collision frames presented to you except late collisions, which indicate serious LAN problems. When the data buffer has been filled, the Ethernet controller clears the E bit in the RX buffer descriptor and generates an interrupt if the I bit is set. If the incoming frame exceeds the length of the data buffer, the Ethernet controller fetches the next RX buffer descriptor in the table and, if it is empty, continues transferring the rest of the frame to this buffer descriptor associated data buffer. The RX buffer descriptor length is determined in the MRBLR value in the SCC2 parameter RAM. You should program the MRBLR to be at least 64 bytes.

During reception, the Ethernet controller checks for a frame that is either too short or too long. When the frame ends, the receive CRC field is checked and written to the data buffer. The data length written to the last buffer descriptor in the Ethernet frame is the length of the entire frame and it enables the software to correctly recognize the frame-too-long condition.

The Ethernet controller then sets the L bit in the RX buffer descriptor, writes the other frame status bits into the RX buffer descriptor, and clears the E bit. Then it generates a maskable interrupt, which indicates that a frame has been received and is in memory. The Ethernet controller then waits for a new frame. It receives serial data least-significant bit first.

16.9.22.7 SCC2 ETHERNET PARAMETER RAM MEMORY MAP. When configured to operate in Ethernet mode, the SCC2 overlays the structure used in Table 16-24 onto the parameters described in Table 16-30.

Table 16-30. SCC2 Ethernet Parameter RAM Memory Map

ADDRESS	NAME	WIDTH	DESCRIPTION
SCC2 Base + 30	**C_PRES**	Word	Preset CRC
SCC2 Base + 34	**C_MASK**	Word	Constant MASK for CRC
SCC2 Base + 38	**CRCEC**	Word	CRC Error Counter
SCC2 Base + 3C	**ALEC**	Word	Alignment Error Counter
SCC2 Base + 40	**DISFC**	Word	Discard Frame Counter
SCC2 Base + 44	**PADS**	Half-word	Short Frame PAD character
SCC2 Base + 46	**RET_LIM**	Half-word	Retry Limit Threshold
SCC2 Base + 48	RET_CNT	Half-word	Retry Limit Counter
SCC2 Base + 4A	**MFLR**	Half-word	Maximum Frame Length Register
SCC2 Base + 4C	**MINFLR**	Half-word	Minimum Frame Length Register
SCC2 Base + 4E	**MAXD1**	Half-word	Max DMA1 Length Register
SCC2 Base + 50	**MAXD2**	Half-word	Max DMA2 Length Register
SCC2 Base + 52	MAXD	Half-word	RX Max DMA
SCC2 Base + 54	DMA_CNT	Half-word	RX DMA Counter

Table 16-30. SCC2 Ethernet Parameter RAM Memory Map (Continued)

ADDRESS	NAME	WIDTH	DESCRIPTION
SCC2 Base + 56	MAX_B	Half-word	Max BD Byte Count
SCC2 Base + 58	**GADDR1**	Half-word	Group Address Filter 1
SCC2 Base + 5A	**GADDR2**	Half-word	Group Address Filter 2
SCC2 Base + 5C	**GADDR3**	Half-word	Group Address Filter 3
SCC2 Base + 5E	**GADDR4**	Half-word	Group Address Filter 4
SCC2 Base + 60	TBUF0_DATA0	Word	Save Area 0–Current Frame
SCC2 Base + 64	TBUF0_DATA1	Word	Save Area 1–Current Frame
SCC2 Base + 68	TBUF0_RBA0	Word	Save RBA–Current Frame
SCC2 Base + 6C	TBUF0_CRC	Word	Save CRC–Current Frame
SCC2 Base + 70	TBUF0_BCNT	Half-word	Save BCNT–Current Frame
SCC2 Base + 72	**PADDR1_L***	Half-word	Physical Address 1 (LSB)
SCC2 Base + 74	**PADDR1_M***	Half-word	Physical Address 1
SCC2 Base + 76	**PADDR1_H***	Half-word	Physical Address 1 (MSB)
SCC2 Base + 78	**P_PER**	Half-word	Persistence
SCC2 Base + 7A	RFBD_PTR	Half-word	RX First BD Pointer
SCC2 Base + 7C	TFBD_PTR	Half-word	TX First BD Pointer
SCC2 Base + 7E	TLBD_PTR	Half-word	TX Last BD Pointer
SCC2 Base + 80	TBUF1_DATA0	Word	Save Area 0–Next Frame
SCC2 Base + 84	TBUF1_DATA1	Word	Save Area 1–Next Frame
SCC2 Base + 88	TBUF1_RBA0	Word	Save RBA–Next Frame
SCC2 Base + 8C	TBUF1_CRC	Word	Save CRC–Next Frame
SCC2 Base + 90	TBUF1_BCNT	Half-word	Save BCNT–Next Frame
SCC2 Base + 92	TX_LEN	Half-word	TX Frame Length Counter
SCC2 Base + 94	**IADDR1**	Half-word	Individual Address Filter 1
SCC2 Base + 96	**IADDR2**	Half-word	Individual Address Filter 2
SCC2 Base + 98	**IADDR3**	Half-word	Individual Address Filter 3
SCC2 Base + 9A	**IADDR4**	Half-word	Individual Address Filter 4
SCC2 Base + 9C	BOFF_CNT	Half-word	Backoff Counter
SCC2 Base + 9E	**TADDR_L**	Half-word	Temp Address (LSB)
SCC2 Base + A0	**TADDR_M**	Half-word	Temp Address
SCC2 Base + A2	**TADDR_H**	Half-word	Temp Address (MSB)

NOTE: You are only responsible for initializing the items in bold. SCC2 Base = (IMMR & 0xFFFF0000) + 0x3D00.
* The bytes inside each half-word are reversed.
All references to registers in the parameter RAM table are actually implemented in the dual-port RAM area as a memory-based register.

- C_PRES—For 32-bit CRC-CCITT, C_PRES should be initialized with 0xFFFFFFFF.
- C_MASK—For 32-bit CRC-CCITT, C_MASK should be initialized with 0xDEBB20E3.
- CRCEC, ALEC, and DISFC—These 32-bit (modulo 2^{32}) counters are maintained by the communication processor module and you can initialize them while the channel is disabled. CRCEC is incremented for each received frame with a CRC error, except it does not include frames not addressed to you, frames received in the out-of-buffers condition, frames with overrun errors, or frames with alignment errors. ALEC is incremented for frames received with dribbling bits, but does not include frames not addressed to you, frames received in the out-of-buffers condition, or frames with overrun errors. DISFC is incremented for frames discarded because of the out-of-buffers condition or an overrun error. The CRC does not have to be correct for this counter to be incremented.
- PADS—You must write the pattern of the pad characters that should be sent when short frame padding is implemented into this 16-bit register. The byte pattern written to the register may be of any value, but both the high and low bytes should be the same.
- RET_LIM—You must write the number of retries that should be made to transmit a frame into this 16-bit register. This value is typically 0xF. If the frame is not transmitted after this limit is reached, an interrupt can be generated.
- RET_CNT is a temporary down-counter used to count the number of retries made.
- MFLR—The SCC2 Ethernet controller checks the length of an incoming Ethernet frame against the user-defined value given in this 16-bit register. Typically this register is set to 0x5EE. If this limit is exceeded, the remainder of the incoming frame is discarded and the LG bit is set in the last RX buffer descriptor belonging to that frame. The SCC2 Ethernet controller reports the frame status and length in the last RX buffer descriptor. MFLR is defined as all the in-frame bytes between the start frame delimiter and the end of the frame.
- MINFLR—The SCC2 Ethernet controller checks the length of an incoming Ethernet frame against the user-defined value given in this 16-bit register that is typically set to 0x40. If the received frame length is less than the register value, then this frame is discarded unless the RSH bit in the PSMR–SCC2 Ethernet is set. If RSH is set, then the SH bit is set in the last RX buffer descriptor belonging to that frame. For transmit operation when the frame is too short, the SCC2 Ethernet controller adds PADs to the transmitted frame, depending on how the PAD bit is set in the TX buffer descriptor and the PAD value in the parameter RAM. Pad characters are added to make the transmit frame MINFLR bytes in length.
- MAXD1—This parameter gives you the option to stop system bus writes from occurring after a frame has exceeded a certain size. However, the value of this register is valid only if an address match is found. The SCC2 Ethernet controller checks the length of an incoming Ethernet frame against the user-defined value given in this 16-bit register that is usually set to 0x5F0. If this limit is exceeded, the remainder of the incoming frame is discarded. The SCC2 Ethernet controller waits to the end of the frame or until MFLR bytes have been received and reports the frame status and the frame length in the last RX buffer descriptor.

- MAXD2—This parameter also gives you the option to stop system bus writes from occurring after a frame has exceeded a certain size. However, the value of this register is valid in promiscuous mode when no address match is detected. The SCC2 Ethernet controller checks the length of an incoming Ethernet frame against the user-defined value given in this 16-bit register that is usually set to 0x5F0. If this limit is exceeded, the remainder of the incoming frame is discarded. The SCC2 Ethernet controller waits until the end of the frame or until MFLR bytes have been received and reports the frame status and length in the last RX buffer descriptor. In a monitor station, MAXD2 can be programmed to a value much less than MAXD1 to receive entire frames addressed to this station, but receives only the headers of the other frames.
- MAXD—For internal use only.
- MAX_B—For internal use only.
- DMA_CNT is a temporary down-counter used to track the frame length.
- GADDR1–4—These four registers are used in the hash table function of the group addressing mode. You can write zeros to these values after reset and before the Ethernet channel is enabled to disable all group hash address recognition functions. The **SET GROUP ADDRESS** command is used to enable the hash table.
- TBUF0_DATA0—For internal use only.
- TBUF0_DATA1—For internal use only.
- TBUF0_RBA0—For internal use only.
- TBUF0_CRC—For internal use only.
- TBUF0_BCNT—For internal use only.
- PADDR1—You must write the 48-bit individual address of this station into this location. PADDR1_L is the lowest order half-word and PADDR1_H is the highest order half-word. PADDR_M is the middle half-word.
- P_PER—This parameter allows the SCC2 Ethernet controller to be less aggressive after a collision. Normally, this parameter should be set to 0x0000. To decrease the aggressiveness of the SCC2 Ethernet controller, you can set P_PER to a value between 1 and 9, with 9 being the least aggressive. The P_PER value is added to the retry count in the backoff algorithm to reduce the probability of transmission on the next time slot.

Note: Using P_PER is fully allowed in the Ethernet/802.3 specifications. In a heavily congested Ethernet LAN, a less aggressive backoff algorithm used by multiple stations on the LAN increases the overall LAN throughput by reducing the probability of collisions. The SBT bit in the PSMR–SCC2 Ethernet offers another way to reduce the aggressiveness of the SCC2 Ethernet controller.

- RFBD_PTR—For internal use only. This register contains the first RX buffer descriptor pointer.
- TFBD_PTR—For internal use only. This register contains the first TX buffer descriptor pointer.
- TLBD_PTR—For internal use only. This register contains the last TX buffer descriptor pointer.
- TX_LEN—For internal use only.
- IADDR1–4 —These four registers are used in the hash table function of the individual addressing mode. You can write zeros to these values after reset and before the Ethernet channel is enabled to disable all individual hash address recognition functions. The **SET GROUP ADDRESS** command is used to enable the hash table.
- BOFF_CNT—For internal use only.
- TADDR—This parameter allows you to add and delete addresses from the individual and group hash tables. After placing an address in TADDR, you must issue the **SET GROUP ADDRESS** command. TADDR_L is the lowest order half-word and TADDR_H is the highest order half-word.TADDR_M is the middle half-word.

16.9.22.8 CONFIGURING THE SCC2 ETHERNET PARAMETERS. You configure the SCC2 to operate as an SCC2 Ethernet controller by setting the MODE field in the general SCC2 mode low register (GSMR_L). The receive errors are reported in the RX buffer descriptor and the transmit errors are reported in the TX buffer descriptor. Several fields in the GSMR_H and GSMR_L must be programmed to special values for Ethernet. You should program the data synchronization register (DSR) using the table below. The GSMR_L programs the first six bytes of the preamble and the SYN1 field of the DSR programs the seventh byte (ox55) of the preamble. Program the 1-byte start delimiter with the value 0xD5 in the SYN2 field of the DSR. Refer to **Section 16.9.2 The General SCC2 Mode Registers** for more information.

DSR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	SYN2								SYN1							
RESET	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1	0
R/W	R/W								R/W							
ADDR	(IMMR & 0xFFFF0000) + 0xA2E															

SYN2—Synchronization 2

This field represents the start frame delimiter for an Ethernet frame. You must set this field to 0xD5.

SYN1—Synchronization 1

This field represents the seventh byte of the preamble for the Ethernet frame. You must set this field to 0x55.

16.9.22.9 SCC2 ETHERNET COMMANDS. You can program the CPM command register (CPCR) with the following commands to transmit data. See **Section 16.2.7.1 CPM Command Register** for additional information.

Note: Before issuing a CPM reset (RST bit in the CPCR), configure the TENA pin as an input.

- **STOP TRANSMIT**—When used with the SCC2 Ethernet controller, this command violates a specific behavior of an Ethernet/IEEE 802.3 station. It should not be used.
- **GRACEFUL STOP TRANSMIT**—This command is used to ensure that transmission stops smoothly after the current frame finishes or undergoes a collision. The GRA bit in the SCCE–Ethernet register is set once transmission stops. Then you can modify the Ethernet transmit parameters and their buffer descriptors. The TBPTR points to the next TX buffer descriptor in the table. Transmission begins once the R bit of the next buffer descriptor is set and the **RESTART TRANSMIT** command is issued.

Note: If the **GRACEFUL STOP TRANSMIT** command is issued and the current transmit frame ends in a collision, the TBPTR points to the beginning of the collided frame with the R bit still set in the TX buffer descriptor. The frame will look as if it was never transmitted.

- **RESTART TRANSMIT**—This command enables the transmission of characters on the transmit channel. The SCC2 Ethernet controller expects it after a **GRACEFUL STOP TRANSMIT** command is issued or a transmitter error occurs. The SCC2 Ethernet controller resumes transmission from the current TBPTR in the channel TX buffer descriptor table.
- **INIT TX PARAMETERS**—This command initializes all the transmit parameters in this serial channel parameter RAM to their reset state. It should only be issued when the transmitter is disabled. The **INIT TX AND RX PARAMS** command can also be used to reset the transmit and receive parameters.

You can program the CPM command register with the following commands to receive data. See **Section 16.2.7.1 CPM Command Register** for additional information.

- **ENTER HUNT MODE**—After the hardware or software is reset and the channel in the SCCM–Ethernet register is enabled, the channel is in receive enable mode and uses the first buffer descriptor in the table. This command is generally used to force the Ethernet receiver to stop receiving the current frame and enter hunt mode. In this mode, the SCC2 Ethernet controller continually scans the input datastream for a transition of carrier sense from inactive to active and then a preamble sequence followed by the start frame delimiter. After receiving the command, the current receive buffer is closed and the CRC calculation is reset. The next RX buffer descriptor is used to receive more frames.

- **CLOSE RX BD**—This command should not be used when the SCC2 is in Ethernet mode.
- **INIT RX PARAMETERS**—This command initializes all the receive parameters in this serial channel parameter RAM to their reset state and should only be issued when the receiver is disabled. The **INIT TX AND RX PARAMS** command can also be used to reset the receive and transmit parameters.
- **SET GROUP ADDRESS**—This command is used to set a bit in one of the 64 bits of the four individual/group address hash filter registers. The individual or group address to be added to the hash table should be written to TADDR_L, TADDR_M, and TADDR_H in the parameter RAM before executing this command. The RISC microcontroller checks the I/G bit in the address stored in TADDR to determine whether to use the individual hash table or the group hash table. A zero in the I/G bit implies an individual address and a 1 in the I/G bit implies a group address. This command can be executed at any time, regardless of whether the Ethernet channel is enabled.

  If you need to delete an address from the hash table, disable the Ethernet channel, clear the hash table registers, and execute this command for the remaining addresses. You must do this because the hash table might have mapped multiple addresses to the same hash table bit.

16.9.22.10 SCC2 ETHERNET ADDRESS RECOGNITION. The SCC2 Ethernet controller can filter the received frames based on different addressing types—physical (individual), group (multicast), broadcast (all-ones group address), and promiscuous. You can set the promiscuous address type in the PSMR–SCC2 Ethernet register. The difference between an individual address and a group address is determined by the I/G bit in the destination address field, which is part of the standard Ethernet protocol. A flowchart for address recognition on received frames is illustrated in Figure 16-106.

In the physical type of address recognition, the SCC2 Ethernet controller compares the destination address field of the received frame with the physical address that you program in PADDR1_H, PADDR1_M, and PADDR1_L. You can also perform address recognition on multiple individual addresses using the IADDR1–4 hash table.

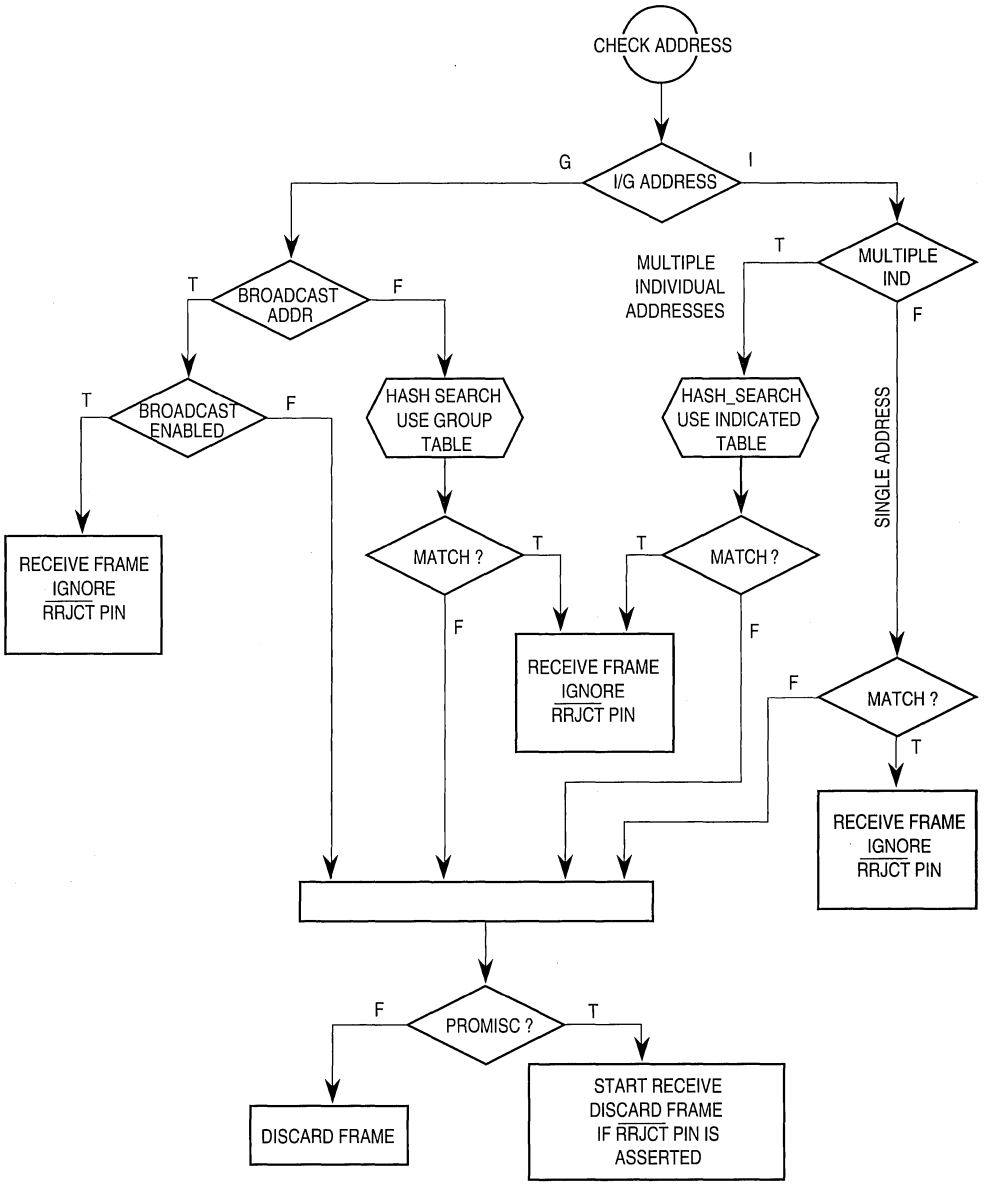


Figure 16-106. Ethernet Address Recognition Flowchart

In the group type of address recognition, the SCC2 Ethernet controller determines whether or not the group address is a broadcast address. If broadcast addresses are enabled, then the frame is accepted, but if the group address is not a broadcast address, then you can perform address recognition on multiple group addresses using the GADDR1–4 hash table. In promiscuous mode, the SCC2 Ethernet controller receives all the incoming frames regardless of their address.

16.9.22.11 HASH TABLE ALGORITHM. Individual and group hash filtering can be operated using a certain process. The SCC2 Ethernet controller maps any 48-bit address into one of 64 bins, which are represented by 64 bits stored in GADDR1–4 or IADDR1–4. When the **SET GROUP ADDRESS** command is executed, the SCC2 Ethernet controller maps the selected 48-bit address into one of the 64 bits by passing the 48-bit address through the on-chip 32-bit CRC generator and selecting 6 bits of the CRC-encoded result to generate a number between 1 and 64. Bits 31–30 of the CRC result select one of the four GADDRs or IADDRs and bits 29–26 of the CRC result select the bit within the selected register.

When the SCC2 Ethernet controller receives a frame, the same process is used. If the CRC generator selects a bit that is set in the group/individual hash table, the frame is accepted. Otherwise, it is rejected. The result is that if eight group addresses are stored in the hash table and random group addresses are received, the hash table prevents roughly 56/64 (87.5%) of the group address frames from reaching memory. Those that do reach memory must be further filtered by the processor to determine if they truly contain one of the eight preferred addresses.

Better performance is achieved by using the group and individual hash tables simultaneously. For instance, if eight group and eight physical addresses are stored in their respective hash tables, 87.5% of all frames are prevented from reaching memory. The effectiveness of the hash table declines as the number of addresses increases. For instance, with 128 addresses stored in a 64-bin hash table, the vast majority of the hash table bits are set, thus preventing a small fraction of the frames from reaching memory.

Note: The hash tables cannot be used to reject frames that match a set of entered addresses because unintended addresses are matched to the same bit in the hash table.

16.9.22.12 INTERPACKET GAP TIME. The minimum interpacket gap time for back-to-back transmission is 9.6µs. The receiver receives back-to-back frames with this minimum spacing. In addition, after the backoff algorithm, the transmitter waits for carrier sense to be negated before retransmitting the frame. Retransmission begins 9.6µs after carrier sense is negated if it stays negated for at least 6.4µs.

16.9.22.13 HANDLING COLLISIONS. If a collision occurs while a frame is being transmitted, the SCC2 Ethernet controller continues transmitting for at least 32 bit times, thus transmitting a JAM pattern that consists of 32 ones. If the collision occurs during the preamble sequence, the JAM pattern will be sent at the end of the preamble sequence.

If a collision occurs within 64 byte times, the retry process is initiated. The transmitter waits a random number of slot times (512 bit times or 52µs). If a collision occurs after 64 byte times, then no retransmission is performed and the buffer is closed with an LC error indication. If a collision occurs while a frame is being received, reception stops. This error is only reported in the buffer descriptor if the length of this frame is greater than or equal to the MINFLR or if the RSH mode is enabled in the PSMR–SCC2 Ethernet.

16.9.22.14 LOOPBACK AND FULL-DUPLEX OPERATION. Both internal and external loopback is supported by the SCC2 Ethernet controller. In loopback mode, both of the SCC2 FIFOs are used and the channel actually operates in a full-duplex fashion. Both internal and external loopback are configured using combinations of the LPB bit in the PSMR–SCC2 Ethernet and the DIAG field in the GSMR_L.

Internal loopback disconnects the serial communication controller from the serial interface. The receive data is connected to the transmit data and the receive clock is connected to the transmit clock. Both FIFOs are used. The transmitted data from the transmit FIFO is received immediately into the receive FIFO. There is no heartbeat check in this mode. TENA should be configured as a general-purpose output, and the HBC bit in the PSMR–SCC2 Ethernet should be zero.

In external loopback operation, the SCC2 Ethernet controller listens for data being received from the EEST at the same time that it is transmitting.

16.9.22.15 SCC2 ETHERNET CONTROLLER ERRORS. The SCC2 Ethernet controller reports frame reception and transmission error conditions using the channel buffer descriptors, the error counters, and the SCCE–Ethernet register. The following transmission errors can be detected by the SCC2 Ethernet controller.

- Transmitter Underrun Error—If this error occurs, the channel sends 32 bits that ensures a CRC error, stops transmitting the buffer, closes it, sets the UN bit in the TX buffer descriptor, and sets TXE in the SCCE–Ethernet register. The channel resumes transmission after it receives the **RESTART TRANSMIT** command.
- Carrier Sense Lost During Frame Transmission Error—When this error occurs and no collision is found in the frame, the channel sets the CSL bit in the TX buffer descriptor, sets the TXE in the SCCE–Ethernet register, and continues the buffer transmission as normal. No retries are performed after this error occurs.
- Retransmission Attempts Limit Expired Error—When this error occurs, the channel stops transmitting the buffer, closes it, sets the RL bit in the TX buffer descriptor, and sets TXE. The channel resumes transmission after it receives the **RESTART TRANSMIT** command.
- Late Collision Error—When this error occurs, the channel stops transmitting the buffer, closes it, sets the LC bit in the TX buffer descriptor, and sets TXE. The channel resumes transmission after it receives the **RESTART TRANSMIT** command. This error is discussed further in the definition of the LCW bit in the PSMR–SCC2 Ethernet.
- Heartbeat Error—Some transceivers have a self-test feature called "heartbeat" or "signal quality error." To signify a good self-test, the transceiver indicates a collision to the MPC823 within 20 clocks after the Ethernet controller transmits a frame. This indication does not imply a real collision error on the network, but is rather an indication that the transceiver still seems to be functioning properly. This is called the heartbeat condition.

  If the HBC bit is set in the PSMR–SCC2 Ethernet and the MPC823 does not detect a heartbeat condition after transmitting a frame, then a heartbeat error occurs. In which case, the channel closes the buffer, sets the HB bit in the TX buffer descriptor, and generates the TXE interrupt if it is enabled.

The following reception errors can be detected by the SCC2 Ethernet controller:

- Overrun Error—The SCC2 Ethernet controller maintains an internal FIFO for receiving data. If a receiver FIFO overrun occurs, the channel writes the received data byte to the internal FIFO over the previously received byte. The previous data byte and frame status are lost. The channel closes the buffer, sets the OV bit in the RX buffer descriptor, RXF in the SCCE–Ethernet register, and increments the discarded frame counter. The receiver then enters hunt mode.
- Busy Error—This error occurs when a frame has been received and discarded because of a lack of buffers. The channel sets the BSY bit in the SCCE–Ethernet register and increments the discarded frame counter.

- Non-Octet Error (Dribbling Bits)—The SCC2 Ethernet controller handles up to seven dribbling bits when the receive frame terminates nonoctet aligned and it checks the CRC of the frame on the last octet boundary. If there is a CRC error, then the frame nonoctet aligned error is reported, the RXF bit is set, and the alignment error counter is incremented. If there is no CRC error, then no error is reported.
- CRC Error—When a CRC error occurs, the channel closes the buffer, sets the CR bit in the RX buffer descriptor, and the RXF bit in the SCCE–Ethernet register. The channel also increments the CRC error counter (CRCEC). After receiving a frame with a CRC error, the receiver enters hunt mode. CRC checking cannot be disabled, but the CRC error can be ignored if checking is not required.

16.9.23 Programming the SCC2 Ethernet Controller

16.9.23.1 SCC2 ETHERNET MODE REGISTER. When the SCC2 is in Ethernet mode, the 16-bit, memory-mapped, read/write protocol-specific mode register is referred to as the SCC2 Ethernet mode register (PSMR–SCC2 Ethernet). Since each protocol has specific requirements, the PSMR bits are different for each implementation.

PSMR–SCC2 ETHERNET

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	HBC	FC	RSH	IAM	CRC		PRO	BRO	SBT	LPB	RES	LCW	NIB			FDE
RESET	0	0	0	0	0		0	0	0	0	0	0	0			0
R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W			R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA28															

HBC—Heartbeat Checking

0 = No heartbeat checking is performed. Do not wait for a collision after transmission.
1 = Wait 20 transmit clocks or 2μs for a collision asserted by the transceiver after transmission. The HB bit in the TX buffer descriptor is set if the heartbeat is not heard within 20 transmit clocks.

FC—Force Collision

0 = Normal operation.
1 = The channel forces a collision when each frame is transmitted. The MPC823 should be configured in loopback operation when using this feature so that you can test the collision logic. In the end, the retry limit for each transmit frame is exceeded.

RSH—Receive Short Frames

0 = Discard short frames that are not as long as MINFLR.
1 = Receive short frames.

IAM—Individual Address Mode

0 = Normal operation. A single 48-bit physical address that is stored in PADDR1_x is checked when it is received.
1 = The individual hash table is used to check all individual addresses that are received.

CRC—CRC Selection

00 = Reserved.
01 = Reserved.
10 = 32-bit CCITT-CRC (Ethernet). X32 + X26 + X23 + X22 + X16 + X12 + X11 + X10 + X8 + X7 + X5 + X4 + X2 + X1 +1. Select this to comply with Ethernet specifications.
11 = Reserved.

PRO—Promiscuous

0 = Check the destination address of the incoming frames.
1 = Receive the frame regardless of its address.

BRO—Broadcast Address

0 = Receive all frames containing the broadcast address.
1 = Reject all frames containing the broadcast address, unless the PRO bit = 1.

SBT—Stop Backoff Timer

0 = The backoff timer is functioning normally.
1 = The backoff timer for the random wait after a collision is stopped when carrier sense is active. This method makes the retransmission less aggressive than the maximum allowed in the IEEE 802.3 standard. The persistence (P_Per) parameter in the parameter RAM can be used in combination with, or in place of, the SBT bit.

LPB—Loopback Operation

0 = Normal operation.
1 = The channel is configured into internal or external loopback operation as determined by the DIAG field of the GSMR_L. For external loopback, the DIAG field should be configured for normal operation and for internal loopback they should be configured for loopback operation.

Bit 10—Reserved

This bit is reserved and should be set to 0.

LCW—Late Collision Window

0 = A late collision is any collision that occurs at least 64 bytes from the preamble.
1 = A late collision is any collision that occurs at least 56 bytes from the preamble.

NIB—Number of Ignored Bits

This parameter determines how soon after RENA assertion the SCC2 Ethernet controller should begin looking for the start frame delimiter. In most situations, you should select 22 bits.

000 = Begin searching for the SFD 13 bits after the assertion of RENA.
001 = Begin searching for the SFD 14 bits after the assertion of RENA.
010 = Begin searching for the SFD 15 bits after the assertion of RENA.
011 = Begin searching for the SFD 16 bits after the assertion of RENA.
100 = Begin searching for the SFD 21 bits after the assertion of RENA.
101 = Begin searching for the SFD 22 bits after the assertion of RENA.
110 = Begin searching for the SFD 23 bits after the assertion of RENA.
111 = Begin searching for the SFD 24 bits after the assertion of RENA.

FDE—Full-Duplex Ethernet

0 = Disable full-duplex Ethernet mode.
1 = Enable full-duplex Ethernet mode.

Note: When this bit is set to 1, you must also set the LPB bit to 1.

16.9.23.2 SCC2 ETHERNET RECEIVE BUFFER DESCRIPTOR. The SCC2 Ethernet controller uses the receive (RX) buffer descriptor to report information about the received data for each buffer. Figure 16-107 illustrates an Ethernet receive buffer descriptor example.

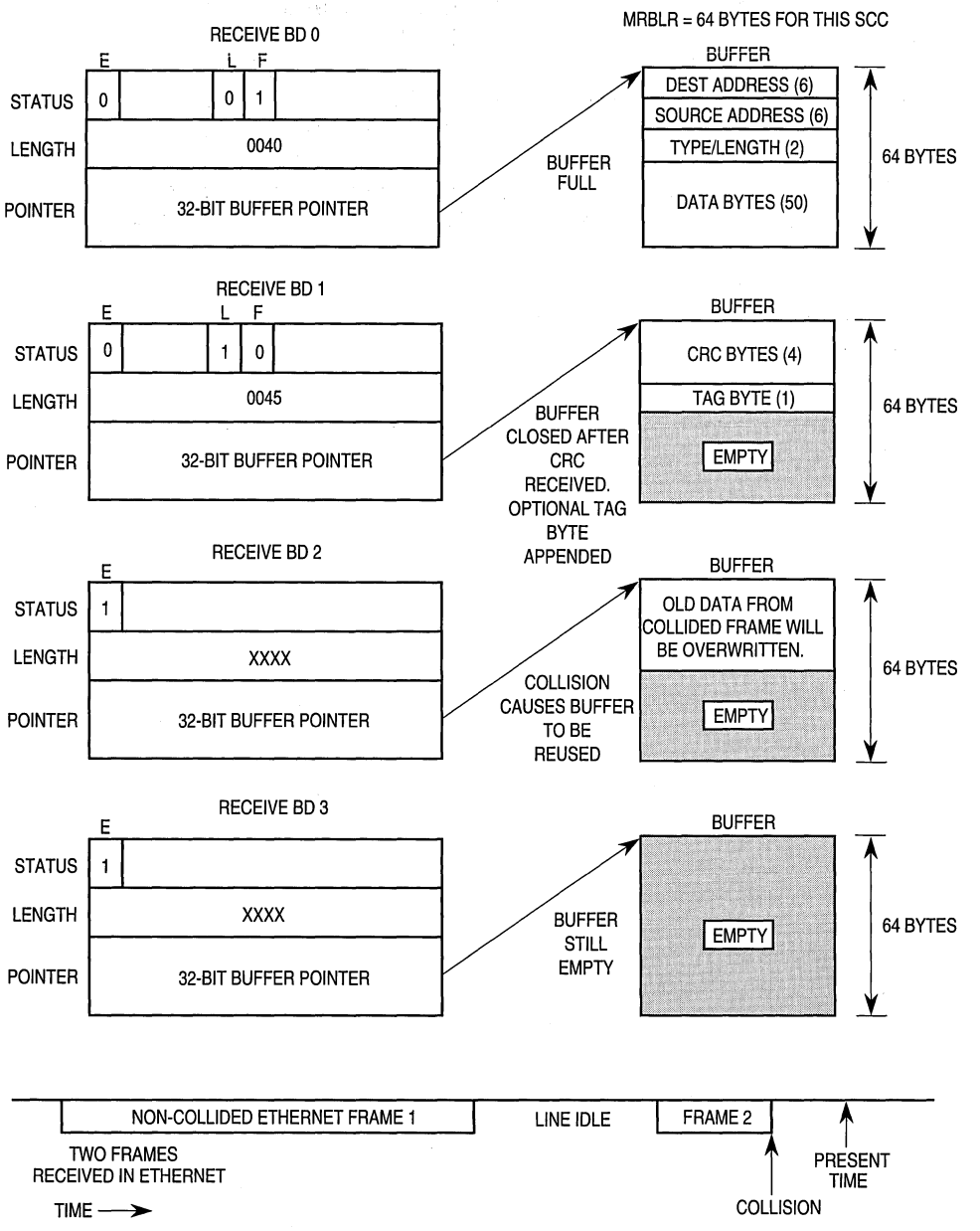


Figure 16-107. Ethernet Receive Buffer Descriptor Example

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET+0	**E**	RES	**W**	**I**	L	F	RES	M	RES		LG	NO	SH	CR	OV	CL
OFFSET+2	DATA LENGTH															
OFFSET+4	**RX DATA BUFFER POINTER**															
OFFSET+6																

NOTE: You are only responsible for initializing the items in bold.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

E—Empty

0 = The data buffer associated with this RX buffer descriptor has been filled with data or has stopped receiving data because an error occurred. The core is free to examine or write to any fields of this RX buffer descriptor. The communication processor module does not use this buffer descriptor as long as the E bit is zero.

1 = The data buffer associated with this RX buffer descriptor is empty or is currently receiving data. This RX buffer descriptor and its associated receive buffer are owned by the communication processor module. Once the E bit is set, the core should not write any fields of this RX buffer descriptor.

Bits 1, 6, and 8–9—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the RX buffer descriptor table.

1 = This is the last buffer descriptor in the RX buffer descriptor table. After this buffer is used, the communication processor module receives incoming data into the first buffer descriptor that RBASE points to in the table. The number of RX buffer descriptors in this table is programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is used.

1 = The RXB or RXF bit of the SCCE–Ethernet register is set when this buffer is used by the SCC2 Ethernet controller. These two bits can cause interrupts if they are enabled.

L—Last in Frame

The Ethernet controller sets this bit when this buffer is the last one in a frame. If the end of a frame is reached or an error is received, one or more of the CL, OV, CR, SH, NO, and LG bits are set. The SCC2 Ethernet controller writes the number of frame octets to the DATA LENGTH field.

0 = The buffer is not the last one in a frame.
1 = The buffer is the last one in a frame.

F—First in Frame

The SCC2 Ethernet controller sets this bit when this buffer is the first one in a frame.

0 = The buffer is not the first one in a frame.
1 = The buffer is the first one in a frame.

M—Miss

The SCC2 Ethernet controller sets this bit for frames that are accepted in promiscuous mode, but are flagged as a "miss" by the internal address recognition. Thus, while in promiscuous mode, you can use the this bit to determine whether the frame is destined to this station. This bit is valid only if the L bit is set.

0 = The frame is received because of an address recognition hit.
1 = The frame is received because of promiscuous mode.

LG—RX Frame Length Violation

This bit indicates that a frame length greater than the maximum defined for this channel has been recognized. Only the maximum-allowed number of bytes is written to the data buffer.

NO—RX Nonoctet Aligned Frame

This bit indicates that a frame containing a number of bits not divisible by eight is received. Also, the CRC check that occurs at the preceding byte boundary has generated an error.

SH—Short Frame

This bit indicates that a frame length less than the minimum defined for this channel has been recognized. This can only happen if the RSH bit is set in the PSMR–SCC2 Ethernet.

CR—RX CRC Error

This bit indicates that this frame contains a CRC error.

OV—Overrun

This bit indicates that a receiver overrun has occurred during frame reception.

CL—Collision

This bit indicates that this frame is closed because a collision has occurred during frame reception. This bit is only set if a late collision occurs or if the RSH bit is enabled in the PSMR–SCC2 Ethernet. Late collisions are better defined in the LCW bit of the same register.

DATA LENGTH

These bits represent the number of octets the communication processor module writes into this buffer descriptor data buffer. It is written by the communication processor module once the buffer is closed. When this buffer descriptor is the last buffer descriptor in the frame, the DATA LENGTH contains the total number of frame octets (including four bytes for CRC).

Note: The actual amount of memory allocated for this buffer should be greater than or equal to the contents of the MRBLR.

RX DATA BUFFER POINTER

These bits always point to the first location of the associated data buffer, can reside in internal or external memory, and must be divisible by four.

16.9.23.3 SCC2 ETHERNET TRANSMIT BUFFER DESCRIPTOR. Data is sent to the SCC2 Ethernet controller for transmission on an SCC2 channel by arranging it in buffers referenced by the channel's transmit (TX) buffer descriptor table. Using the buffer descriptors, the SCC2 Ethernet controller confirms transmission or indicates error conditions so that the core knows the buffers have been serviced.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	R	PAD	W	I	L	TC	DEF	HB	LC	RL	RC				UN	CSL
OFFSET + 2	**DATA LENGTH**															
OFFSET + 4	**TX DATA BUFFER POINTER**															
OFFSET + 6																

NOTE: You are only responsible for initializing the items in bold.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

R—Ready

0 = The data buffer associated with this buffer descriptor is not ready for transmission and you are free to manipulate this buffer descriptor or its associated data buffer. The communication processor module clears this bit after the buffer has been transmitted or after an error condition is encountered.

1 = The data buffer, which you have prepared for transmission, has not been transmitted or is currently being transmitted. You cannot write any fields of this buffer descriptor once this bit is set.

PAD—Short Frame Padding

This bit is only valid when the L bit is set. Otherwise, it is ignored.

0 = Do not add pads to short frames.

1 = Add pads to short frames. Pad bytes are inserted until the length of the transmitted frame equals the MINFLR and they are stored in pads in the parameter RAM.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the TX buffer descriptor table.

1 = This is the last buffer descriptor in the TX buffer descriptor table. After this buffer is used, the communication processor module receives incoming data into the first buffer descriptor that TBASE points to in the table. The number of TX buffer descriptors in this table is programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

Note: The TX buffer descriptor table must contain more than one buffer descriptor in SCC2 Ethernet mode.

I—Interrupt

0 = No interrupt is generated after this buffer is serviced.

1 = The TXB or TXE bit is set in the SCCE–Ethernet register after this buffer is serviced. These bits can cause interrupts if they are enabled.

L—Last

0 = This is not the last buffer in the transmit frame.

1 = This is the last buffer in the transmit frame.

TC—TX CRC

This bit is valid only when the L bit is set. Otherwise, it is ignored.

0 = End transmission immediately after the last data byte.

1 = Transmit the CRC sequence after the last data byte.

DEF—Defer Indication

This bit indicates that this frame had a collision before it was sent. It is useful for channel statistics. The SCC2 Ethernet controller writes this bit after it finishes transmitting the associated data buffer.

HB—Heartbeat

This bit indicates the collision input was not asserted within 20 transmit clocks after transmission. It cannot be set unless the HBC bit is set in the PSMR–SCC2 Ethernet. The SCC2 Ethernet controller writes this bit after it finishes transmitting the associated data buffer.

LC—Late Collision

This bit indicates that a collision occurred after the number of bytes defined for the LCW bit in the PSMR–SCC2 Ethernet are transmitted. The SCC2 Ethernet controller stops transmitting and writes this bit after it finishes transmitting the associated data buffer.

RL—Retransmission Limit

This bit indicates when the transmitter fails Retry Limit + 1 attempts to successfully transmit a message because of repeated collisions on the medium. The SCC2 Ethernet controller writes this bit after it finishes transmitting the associated data buffer.

RC—Retry Count

These bits indicate the number of retries required before this frame is successfully transmitted. If RC = 0, then the frame is transmitted correctly the first time. If RC = 15 and RET_Lim = 15 in the parameter RAM, then 15 retries are required. If RC = 15 and RET_Lim > 15 in the parameter RAM, then 15 or more retries are required. The SCC2 Ethernet controller writes these bits after it finishes transmitting the associated data buffer.

UN—Underrun

This bit indicates that the SCC2 Ethernet controller has encountered a transmitter underrun condition while transmitting the associated data buffer. The SCC2 Ethernet controller writes this bit after it finishes transmitting the associated data buffer.

CSL—Carrier Sense Lost

This bit indicates that the carrier sense was lost during frame transmission. The SCC2 Ethernet controller writes this bit after it finishes transmitting the associated data buffer.

DATA LENGTH

This field represents the number of octets the SCC2 Ethernet controller should transmit from this buffer descriptor data buffer. It is never modified by the communication processor module. The value of this field should be greater than zero. The SCC2 Ethernet controller writes these bits after it finishes transmitting the associated data buffer.

TX DATA BUFFER POINTER

This field contains the address of the associated data buffer, can be even or odd, and reside in internal or external memory. This value is never modified by the communication processor

module. The SCC2 Ethernet controller writes these bits after it finishes transmitting the associated data buffer.

16.9.23.4 SCC2 ETHERNET EVENT REGISTER. When the SCC2 is in Ethernet mode, the 16-bit memory-mapped SCC2 event register is referred to as the SCC2 Ethernet event register (SCCE–Ethernet). Since each protocol has specific requirements, the SCCE bits are different for each implementation. This register is used to generate interrupts and report events recognized by the Ethernet channel. When an event is recognized, the SCC2 Ethernet controller sets the corresponding bit in the SCCE–Ethernet register. Interrupts generated by this register can be masked in the SCCM–Ethernet register. An example of interrupts that can be generated in the Ethernet protocol is illustrated in Figure 16-108.

A bit is cleared by writing a 1(writing a zero has no effect) and more than one bit can be cleared at a time. All unmasked bits must be cleared before the communication processor module clears the internal interrupt request. This register is cleared at reset and can be read at any time.

SCCE–ETHERNET

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED								GRA	RESERVED		TXE	RXF	BSY	TXB	RXB
RESET	0								0	0		0	0	0	0	0
R/W	R/W								R/W	R/W		R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA30															

Bits 0–7 and 9–10—Reserved

These bits are reserved and should be set to 0.

GRA—Graceful Stop Complete

This bit indicates that a graceful stop, initiated by the **GRACEFUL STOP TRANSMIT** command, is now complete. This bit is set as soon the transmitter finishes any frame that was in progress when the command was issued. It is set immediately if no frame was in progress when the command was issued.

TXE—TX Error

This bit indicates that an error has occurred on the transmitter channel.

RXF—RX Frame

This bit indicates that a complete frame has been received on the Ethernet channel.

BSY—Busy Condition

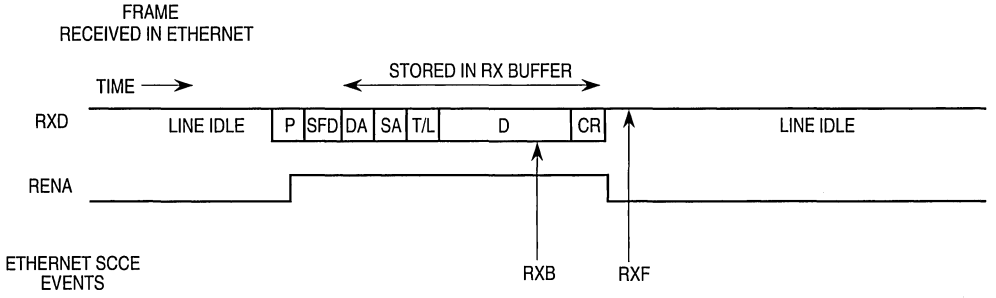
This bit indicates when a frame is received and discarded due to a lack of buffers.

TXB—TX Buffer

This bit indicates that a buffer has been transmitted on the Ethernet channel.

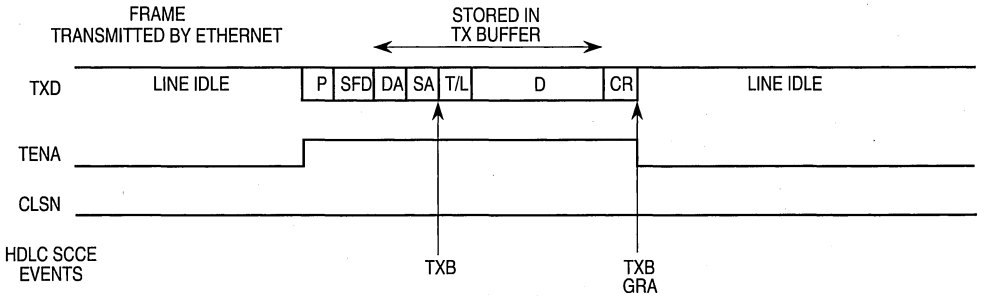
RXB—RX Buffer

This bit indicates that a buffer that was not a complete frame was received on the Ethernet channel.



NOTES:
1. RXB event assumes receive buffers are 64 bytes each.
2. The RENA events, if required, must be programmed in the port C parallel I/O, not in the SCC itself.
3. The RXF interrupt may occur later than RENA due to receive FIFO latency.

LEGEND:
P = Preamble, SFD = Start Frame Delimiter, DA and SA = Source/Destination Address, T/L = Type/Length, D = Data, and CR = CRC bytes.



NOTES:
1. TXB events assume the frame required two transmit buffers.
2. The GRA event assumes a GRACEFUL STOP TRANSMIT command was issued during frame transmission.
3. The TENA or CLSN events, if required, must be programmed in the port C parallel I/O, not in the SCC itself.

Figure 16-108. Ethernet Interrupt Events Example

16.9.23.5 SCC2 ETHERNET MASK REGISTER. When the SCC2 is in Ethernet mode, the 16-bit read/write SCC2 mask register is referred to as the SCC2 Ethernet mask register (SCCM–Ethernet). Since each protocol has specific requirements, the SCCM bits are different for each implementation. This register has the same bit formats as the SCCE–Ethernet register. If a bit in the SCCM–Ethernet register is a 1, the corresponding interrupt in the SCCE–Ethernet register is enabled. If the bit is zero, the corresponding interrupt in the SCCE–Ethernet register is masked.

SCCM–ETHERNET

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED								GRA	RESERVED		TXE	RXF	BSY	TXB	RXB
RESET	0								0	0		0	0	0	0	0
R/W	R/W								R/W	R/W		R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA34															

16.9.23.6 SCC2 ETHERNET STATUS REGISTER. Since all Ethernet mode selections are in the GSMR_x and PSMR registers, the SCC2 Ethernet status register (SCCS–Ethernet) is not used when the SCC2 is in Ethernet mode. The current state of the RENA and CLSN signals can be found in port C.

16.9.23.7 SCC2 ETHERNET PROGRAMMING EXAMPLE. The following is an example initialization sequence for the SCC2 in Ethernet mode. The CLK1 pin is used for the Ethernet receiver and the CLK2 pin is used for the transmitter.

1. Configure the port A pins to enable the TXD1 and RXD1 pins. Write PAPAR bits 12 and 13 with ones, PADIR bits 12 and 13 with zeros, and PAODR bit 13 with zero.
2. Configure the port C pins to enable $\overline{\text{CTS2}}$ (CLSN) and $\overline{\text{CD2}}$ (RENA). Write PCPAR and PCDIR bits 9 and 8 with zeros and PCSO bits 9 and 8 with ones.
3. Do not enable the $\overline{\text{RTS2}}$ (TENA) pin yet because the pin is still functioning as $\overline{\text{RTS}}$ and transmission on the LAN could accidentally begin.
4. Configure port A to enable the CLK1 and CLK2 pins. Write PAPAR bits 7 and 6 with ones and PADIR bits 7 and 6 with zeros.
5. Connect the CLK1 and CLK2 pins to SCC2 using the serial interface. Write the R2CS field in the SICR to 101 and the T2CS field to 100.
6. Connect the SCC2 to the NMSI and clear the SC2 bit in the SICR.
7. Initialize the SDMA configuration register (SDCR) to 0x0001.
8. Write RBASE and TBASE in the SCC2 parameter RAM to point to the RX buffer descriptor and TX buffer descriptor in the dual-port RAM. Assuming one RX buffer descriptor at the beginning of the dual-port RAM and one TX buffer descriptor following that RX buffer descriptor, write RBASE with 0x2000 and TBASE with 0x2008.
9. Program the CPCR to execute the **INIT RX BD PARAMETER** command for this channel.

10. Write RFCR and TFCR with 0x18 for normal operation.
11. Write MRBLR with the maximum number of bytes per receive buffer. For this case, assume 1,520 bytes, so MRBLR = 0x05F0. In this example, the user wants to receive an entire frame into one buffer, so the MRBLR value is chosen to be the first value larger than 1,518 that is evenly divisible by four.
12. Write C_PRES with 0xFFFFFFFF to comply with 32-bit CCITT-CRC.
13. Write C_MASK with 0xDEBB20E3 to comply with 32-bit CCITT-CRC.
14. Clear CRCEC, ALEC, and DISFC for clarity.
15. Write PAD with 0x8888 for the pad value.
16. Write RET_LIM with 0x000F.
17. Write MFLR with 0x05EE to make the maximum frame size 1,518 bytes.
18. Write MINFLR with 0x0040 to make the minimum frame size 64 bytes.
19. Write MAXD1 and MAXD2 with 0x05EE to make the maximum DMA count 1,518 bytes.
20. Clear GADDR1–GADDR4. The group hash table is not used.
21. Write PADDR1_H with 0x0380, PADDR1_M with 0x12E0, and PADDR1_L with 0x5634 to configure the physical address 8003E0123456.
22. Write P_Per with 0x0000. It is not used.
23. Clear IADDR1–IADDR4. The individual hash table is not used.
24. Clear TADDR_H, TADDR_M, and TADDR_L for clarity.
25. Initialize the RX buffer descriptor and assume the RX data buffer is at 0x00001000 in main memory. Write 0xB000 to Rx_BD_Status, 0x0000 to Rx_BD_Length (optional), and 0x00001000 to Rx_BD_Pointer.
26. Initialize the TX buffer descriptor and assume the TX data frame is at 0x00002000 in main memory and contains fourteen 8-bit characters (destination and source addresses plus the type field). Write 0xFC00 to Tx_BD_Status, add PAD to the frame and generate a CRC. Then write 0x000D to Tx_BD_Length and 0x00002000 to Tx_BD_Pointer.
27. Write 0xFFFF to the SCCE–Ethernet to clear any previous events.
28. Write 0x001A to the SCCM–Ethernet to enable the TXE, RXF, and TXB interrupts.
29. Write 0x20000000 to the CIMR so that SCC2 can generate a system interrupt. The CICR should also be initialized.
30. Write 0x00000000 to the GSMR_H to enable normal operation of all modes.
31. Write 0x1088000C to the GSMR_L to configure the $\overline{\text{CTS}}$ (CLSN) and $\overline{\text{CD}}$ (RENA) pins to automatically control transmission and reception (DIAG field) and the Ethernet mode. TCI is set to allow more setup time for the EEST to receive the MPC823 transmit data. TPL and TPP are set for Ethernet requirements. The DPLL is not used with Ethernet. Notice that the transmitter (ENT) and receiver (ENR) have not been enabled yet.
32. Write 0xD555 to the DSR.

33. Set the PSMR–SCC2 Ethernet to 0x0A0A to configure 32-bit CRC, promiscuous mode and begin searching for the start frame delimiter 22 bits after RENA.
34. Enable the TENA pin ($\overline{RTS}$). Since the MODE field of the GSMR_L is written to Ethernet, the TENA signal is low. Write PCPAR bit 14 with a one and PCDIR bit 14 with a zero.
35. Write 0x1088003C to the GSMR_L register to enable the SCC2 transmitter and receiver. This additional write ensures that the ENT and ENR bits are enabled last.

Note: After 14 bytes and the 46 bytes of automatic pad (plus the 4 bytes of CRC) are transmitted, the TX buffer descriptor is closed. Additionally, the receive buffer is closed after a frame is received. Any data received after 1,520 bytes or a single frame causes a busy (out-of-buffers) condition since only one RX buffer descriptor is prepared.

16.10 UNIVERSAL SERIAL BUS CONTROLLER

The universal serial bus (USB) is an industry standard extension to the PC architecture. The USB controller allows the MPC823 to exchange data with a PC host. It supports data exchanges between a host computer and a wide range of simultaneously accessible peripherals. The attached peripherals share USB bandwidth through a host scheduled token-based protocol. The USB physical interconnect is a tiered star topology. A hub is at the center of each star. Each wire segment is a point-to-point connection between the host and a hub or function or a hub connected to another hub/function. There is only one host in any USB system. The USB transfers signal and power over a four-wire cable. The signalling occurs over two wires and point-to-point segments. The USB full-speed signalling bit rate is 12Mbps. A limited capability low-speed signalling mode is also defined at 1.5Mbps.

The MPC823 USB controller consists of a transmitter module, receiver module, and two protocol state machines. The protocol state machines control the receiver and transmitter modules. One state machine implements the function state diagram and the other implements the host state diagram. The MPC823 USB controller is capable of implementing a USB function endpoint, a USB host, or both for testing purposes (loop-back diagnostics). Figure 16-109 illustrates the USB controller block diagram.

For USB implementation, it is recommended that you get a copy of the USB Specification as a supplement to this manual. You can download a copy from http://www.usb.org.

The USB transmitter contains four independent FIFOs, each containing 16 bytes. There is a dedicated FIFO for each of the four supported endpoints. The USB receiver has a single 16-byte FIFO. When the USB controller is not enabled in the USMOD, it consumes minimal power.

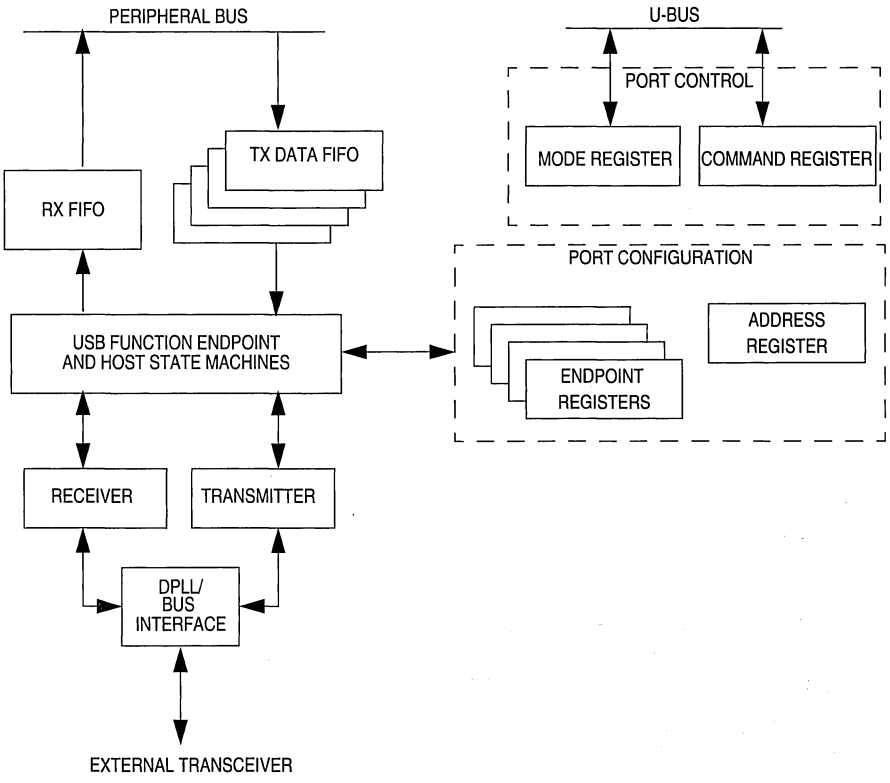


Figure 16-109. USB Controller Block Diagram

16.10.1 Features

The following list summarizes the USB slave mode features:

- Supports USB slave mode
- Four independent endpoints support control, bulk, interrupt and isochronous data transfers
- CRC16/CRC5 generation and checking
- NRZI encoding/decoding with bit stuffing
- 12 or 1.5Mbps data rate
- Flexible data buffers with multiple buffers per frame
- Automatic retransmission upon transmit error

The following list summarizes the USB host controller features:

- Supports control, bulk, interrupt, and isochronous data transfers
- CRC16 generation and checking
- NRZI encoding/decoding with bit stuffing
- 12Mbps data rate
- Flexible data buffers with multiple buffers per frame
- Supports local loopback mode for diagnostics

16.10.2 Host Controller Limitations

The USB host controller only supports full-speed (12Mbps) USB devices. The transmitter does not support preamble generation for low-speed transmissions. The following tasks are not supported by the hardware and should be implemented by software:

- CRC5 generation for token. Since CRC5 is calculated on 11 bits, this should not impose a large CPU overhead.
- Retransmission after an error and error recovery.
- Generation and transmission of SOF token every 1ms.
- Scheduling the various transfers within frame and between frames. The MPC823 USB host controller does not integrate the root hub. An external hub is required when more than one device is connected to the host.
- The host controller programming model is very similar to the function endpoint programming model and does not conform to the Open Host Controller Interface (OHCI). Although Motorola plans to support full host functionality, our implementation will not be OHCI or UHCI (Universal Host Controller Interface) compatible. Those standards define the software programming model so that software drivers can be hardware-independent. Our implementation has a much more simplified software interface.

16.10.3 USB Controller Pin Functions and Clocking

The USB controller interfaces to the USB through a differential line driver and receiver. The OE signal enables the line driver when the USB controller transmits on the bus.

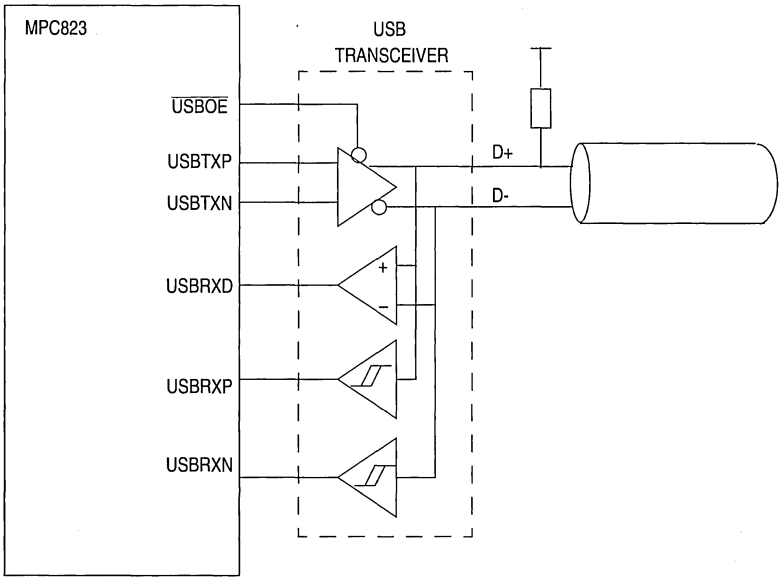


Figure 16-110. USB Interface

There are six I/O port pins associated with the USB port and their functionality is described in Table 16-31. Some transceivers may need additional control lines (speed select or low-power control), which can be supported by the general-purpose output lines.

Table 16-31. USB Pin Functionality

PIN SYMBOL	I/O	FUNCTION
USBTXN, USBTXP	O	Outputs from the USB transmitter, inputs to the differential driver. TP / TN / RESULT 0 / 0 / single ended "0" 0 / 1 / logic "0" 1 / 0 / logic "1" 1 / 1 / n/a
$\overline{\text{USBOE}}$	O	Output Enable. Active low, enables the transceiver to transmit data on the bus.
USBRXD	I	Receive Data. Input the USB receiver from the differential line receiver.
UBRXP, USBRXN	I	Gated version of D+ and D-. Used to detect single-ended zero, and interconnect speed. RP / RN / RESULT 0 / 0 / single ended "0" 1 / 0 / full speed 0 / 1 / low speed 1 / 1 / n/a

The USB controller accepts a reference clock that is four times the USB bit rate. This clock is programmed in the R1CS field of the serial interface clock register (SICR) and used by the DPLL circuitry to recover the bit rate clock. Refer to **Section 16.7.5.3 Serial Interface Clock Route Register** for more information.

Note: The MPC823 can run at different frequencies, but the USB reference clock must be four times the USB bit rate. The reference clock must be 48MHz for a 12Mb full-speed transfer or 6MHz for a 1.5Mb low-speed transfer.

16.10.4 Transmission and Reception Process

After reset, the USB controller is addressable at the default address (0x00). During the enumeration process, the host assigns a unique address to the USB controller. The software should program the USB address register with the assigned address. The USB controller supports four independent endpoints, which can each be configured to support control, interrupt, bulk, or isochronous transfers modes when you program the USB endpoint registers.

Note: You must configure endpoint 0 as a control transfer type. This endpoint is used by the USB system software as a control pipe and any additional control pipes may be provided by other endpoints.

Once enabled, the USB controller looks for valid token packets. Tokens that are not valid (the PID or CRC check fails or the packet length is not 3 bytes) are ignored.

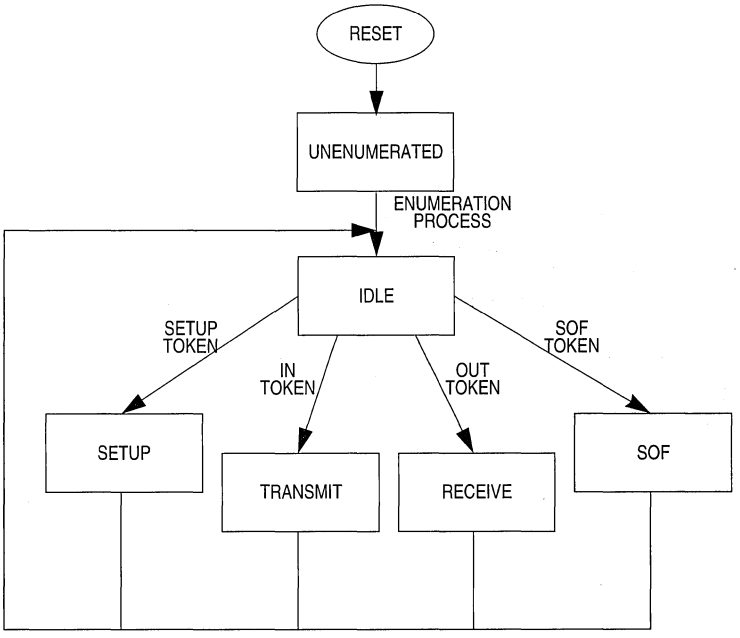


Figure 16-111. USB Controller Operating Modes

16.10.4.1 OUT TOKEN. When the USB controller receives an OUT token, data reception begins. The USB controller fetches the next buffer descriptor associated with the endpoint and if it is empty, it starts transferring the incoming packet to the buffer descriptor's associated data buffer. When the data buffer has been filled, the USB controller clears the E bit in the buffer descriptor and generates an interrupt if the I bit in the buffer descriptor is set. If the incoming packet exceeds the length of the data buffer, the USB controller fetches the next buffer descriptor in the table and, if it is empty, continues transferring the rest of the packet to this buffer descriptor's associated buffer. Otherwise, if it is full, an error occurs. The entire data packet, including the DATA0/DATA1 packet ID (PID), is written to the receive buffers and the PID is reflected in the PID field of the receive buffer descriptor. It is your responsibility to program the driver software to check data packet synchronization by monitoring the DATA0/DATA1 PID sequence toggle.

Table 16-32. USB Out Token Reception

DATA PACKET CORRUPTED	USEPx RHS FIELD	HANDSHAKE SENT TO HOST
Yes	N/A	None (Data Discarded)
No	00 (Normal)	ACK
No	01 (Ignore)	None
No	10 (NAK)	NAK
No	11 (STALL)	STALL

If the packet was received error-free (no CRC errors and no bit stuff error), depending on the endpoint transfer mode configuration, an ACK handshake will be transmitted to the host. If a reception error occurred, no handshake packet is returned, and the error status bits will be set in the last buffer descriptor associated with this packet. If the RHS field in the endpoint's configuration register is programmed to respond with NAK, a NAK handshake is returned instead of ACK. If the RHS field is programmed to respond with STALL, a STALL handshake is returned. In both cases, the buffer will receive the data packet if the buffer descriptors are available.

16.10.4.2 IN TOKEN. To guarantee data transfer, the control software must preload the endpoint FIFO with a data packet prior to receiving an IN token. The software should set up the endpoint transmit buffer descriptor table and set the STR bit in the **USB** command register. The USB controller will fill the transmit FIFO and wait until it receives the IN token. Once it is received, transmission begins.

Table 16-33. USB In Token Reception

DATA PACKET CORRUPTED	USEPx THS FIELD	HANDSHAKE SENT TO HOST
Yes	N/A	None (Data Discarded)
No	00 (Normal)	ACK
No	01 (Ignore)	None
No	10 (NAK)	NAK
No	11 (STALL)	STALL

If data is not ready in the transmit FIFO or if the THS field in the endpoint's configuration register is set to respond with NAK, a NAK handshake is returned. If the THS field in the endpoint's configuration register is set to respond with STALL, a STALL handshake is returned. When the end of the current buffer descriptor has been reached and the last buffer in the packet bit is set, the CRC is appended. Following the transmission of a frame, the USB controller waits for a handshake packet, depending on the configuration of the endpoint. If the host fails to acknowledge the packet, the timeout status bit will be set in the buffer descriptor. It is your responsibility to program the driver software to set the proper DATA0/DATA1 PID in the transmitted packet.

16.10.4.3 SETUP TOKEN. Setup transactions are similar in format to an OUT token, but you should use a SETUP rather than an OUT PID. A SETUP token is only recognized by an endpoint that is configured as a control endpoint. Once the SETUP token is received, setup data reception begins. The USB controller fetches the next buffer descriptor associated with the endpoint, and if it is empty, starts transferring the incoming packet to the buffer descriptor's associated data buffer. When the data buffer has been filled, the USB controller clears the E bit in the buffer descriptor and generates an interrupt if the I bit in the buffer descriptor is set. If the incoming packet exceeds the length of the data buffer, the USB controller fetches the next buffer descriptor in the table and, if it is empty, continues transferring the rest of the packet to this buffer descriptor's associated buffer. If it is full, an error occurs. The entire data packet including the DATA0 PID are written to the receive buffers. If the packet was received error-free (no CRC errors and no bit stuff error) an ACK handshake will be transmitted to the host. If a reception error occurred, no handshake packet will be returned and the error status bits will be set in the last buffer descriptor associated with this packet.

16.10.4.4 SOF TOKEN. When a start of frame (SOF) token packet is received, the USB controller issues a SOF maskable interrupt and the frame number entry in the parameter RAM is updated.

16.10.4.5 PRE TOKEN. The PRE token signals the hub that a low-speed transaction is about to occur. The PRE token is only read by the hub. The USB controller ignores the PRE token function in slave mode. In host mode, the USB controller cannot generate a PRE token.

16.10.5 USB Controller Parameter RAM Memory Map

The USB controller parameter RAM area begins at the USB base address. The area is used for the general USB parameters. Notice that it is similar to the SCC2 general-purpose parameter RAM.

Table 16-34. USB Parameter RAM Memory Map

ADDRESS	NAME	WIDTH	DESCRIPTION
USB Base + 00	**EP0PTR**	Half Word	Endpoint 0 Register
USB Base + 02	**EP1PTR**	Half Word	Endpoint 1 Register
USB Base + 04	**EP2PTR**	Half-Word	Endpoint 2 Register
USB Base + 06	**EP3PTR**	Half Word	Endpoint 3 Register
USB Base + 08	RSTATE	Word	RX Internal State
USB Base + 0C	RPTR	Word	RX Internal Data Pointer
USB Base + 10	**FRAME_N**	Half Word	Frame Number
USB Base + 12	RBCNT	Half-Word	RX Internal Byte Count
USB Base + 14	RTEMP	Word	RX Temp

NOTE: You are only responsible for initializing the items in bold.
USB Base = (IMMR & 0xFFFF0000) + 0x3C00.
All references to registers in the parameter RAM table are actually implemented in the dual-port RAM area as a memory-based register.

You must initialize certain parameter RAM values before the USB controller is enabled. Other values are initialized by the communication processor module. Once initialized, the parameter RAM values do not need to be accessed by your software. They should only be modified when no USB activity is in progress.

- EPxPTR—The endpoint parameters block pointers are index pointers to the endpoint's parameter block. The parameter block can be allocated to any address divisible by 32 in the dual-port RAM. The format of the endpoint parameter block is shown in Table 16-35.

EPxPTR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	ENDPOINT INDEX POINTER											0	0	0	0	0
R/W	R/W											R/W	R/W	R/W	R/W	R/W
RESET	0											0	0	0	0	0
ADDR	USB BASE + 0x00 (EP0PTR), 0x02 (EP1PTR), 0x04 (EP2PTR), 0x06 (EP3PTR)															

- FRAME_N—The frame number entry is updated by the USB controller when a SOF token is received. The entry contains 11 bits representing the frame number. An SOF interrupt is issued when this entry is updated.

FRAME_N

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	V	RESERVED				FRAME NUMBER										
R/W	R/W	R/W				R/W										
RESET	0	0				0										
ADDR	USB BASE + 0x10															

The V bit is set if the SOF token was received error-free.

Table 16-35. Endpoint Parameters Block

ADDRESS	NAME	WIDTH	DESCRIPTION
Base + 00	**RBASE**	Half Word	RX BD Base Address
Base + 02	**TBASE**	Half Word	Tx BD Base Address
Base + 04	**RFCR**	Byte	RX Function Code
Base + 05	**TFCR**	Byte	TX Function Code
Base + 06	**MRBLR**	Half Word	Maximum Receive Buffer Length
Base + 08	**RBPTR**	Half Word	RX BD Pointer
Base + 0a	**TBPTR**	Half Word	TX BD Pointer
Base + 0c	TSTATE	Word	TX Internal State
Base + 10	TPTR	Word	TX Internal Data Pointer
Base + 14	TCRC	Half Word	TX Temp CRC
Base + 16	TBCNT	Half Word	TX Internal Byte Count
Base + 1C	RES	Word	Reserved

NOTE: You are only responsible for initializing the items in bold. Also, Base = (EP x PTR).

- RBASE and TBASE—The receive and transmit buffer descriptor base address entries define the starting point in the dual-port RAM for the set of buffer descriptors to receive and transmit data. This provides a great deal of flexibility in how buffer descriptors for the USB controller are partitioned. By setting the W bit in the last buffer descriptor in each list, you can select how many buffer descriptors to allocate for the transmit and receive side of the USB controller. However, you must initialize these entries before enabling the USB controller. Furthermore, you should not configure buffer descriptor tables of the USB to overlap any other serial channel's buffer descriptors or else erratic operation will occur.

Note: RBASE and TBASE should contain a value that is divisible by 8.

- RFCR and TFCR—The USB function code registers control the value that you would like to appear on the AT pins when the associated SDMA channel accesses memory. It also controls the byte-ordering convention to be used in the transfers.

RFCR AND TFCR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED			BO		AT		

Bits 0–2—Reserved

These bits are reserved and should be set to 0.

BO—Byte Ordering

Set this field to select the required byte ordering for the data buffer. If this field is modified on-the-fly, it will take effect at the beginning of the next frame.

00 = The DEC/Intel convention is used for byte ordering (swapped operation). It is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed as compared to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double word.

1X = Motorola byte ordering (normal operation). It is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT—Address Type 1–3

This field contains the function code value used during this SDMA channel's memory accesses. AT0 will be driven with a one to identify this SDMA channel access as a DMA-type access.

- MRBLR—The maximum receive buffer length register defines the maximum number of bytes that the MPC823 will write to the USB receive buffer before moving to the next buffer. MRBLR must be divisible by 4. The MPC823 can write fewer bytes to the buffer than the MRBLR value if a condition such as an error or end-of-packet occurs, but it will never write more bytes than the MRBLR value. It follows, then, that buffers you supply to the MPC823 should always be at least as long as MRBLR.

  The transmit buffers for the USB channel are not affected in any way by the value programmed into MRBLR. Transmit buffers may be individually chosen to have varying lengths, as needed. The number of bytes to be transmitted is chosen by programming the DATA LENGTH field in the TX buffer descriptor.

Note: MRBLR is not intended to be changed dynamically while the USB channel is operating. However, if it is modified in a single bus cycle with one 16-bit move (not two 8-bit bus cycles back-to-back), then a dynamic change in the receive buffer length can be successfully achieved. This occurs when the communication processor module moves control to the next RX buffer descriptor in the table. Thus, a change to MRBLR will not have an immediate effect. To guarantee the exact RX buffer descriptor on which the change will occur, you should change MRBLR only while the USB receiver is disabled.

- RBPTR—The receiver buffer descriptor pointer points to the next buffer descriptor that the receiver will transfer data to when it is in an idle state or to the current buffer descriptor while processing a frame. RBPTR should be initialized by the software after reset. When the end of the buffer descriptor table is reached, the communication processor module initializes this pointer to the value programmed in the RBASE entry. Although you never need to write RBPTR in most applications (except initialization), you can modify it when the receiver is disabled or when you are sure that no receive buffer is currently in use.
- TBPTR—The transmitter buffer descriptor pointer for each USB endpoint points to the next buffer descriptor that the transmitter will transfer data from when it is in an idle state or to the current buffer descriptor during frame transmission. TBPTR should be initialized by the software after reset. When the end of buffer descriptor table is reached, the communication processor module initializes this pointer to the value programmed in the TBASE entry. Although you never need to write TBPTR in most applications (except initialization), you can modify it when the transmitter is disabled or when you are sure that no transmit buffer is currently in use.
- TSTATE—The transmit internal state entry should be initilaized to zero before enabling the USB controller.
- Other General Parameters—These are the additional parameters listed in Table 16-34 and Table 16-35. You do not need to access these parameters in normal operation and they are only listed because they provide helpful information to experienced users for debugging purposes. The RX and TX internal data pointers (RPTR and TPTR) are updated by the SDMA channels to show the next address in the buffer to be accessed.

  TX internal byte count (TBCNT) is a down-count value that is initialized with the TX buffer descriptor's DATA LENGTH field and decremented with every byte read by the SDMA channels. RX internal byte count (RBCNT)is a down-count value that is initialized with the MRBLR value and decremented with every byte written by the SDMA channels. RSTATE, TSTATE, RTEMP, TTEMP, and the reserved areas are only used by the RISC microcontroller.

16.10.6 USB Commands

You can program the CPM command register (CPCR) with the following commands to transmit data.

USB COMMAND FORMAT (CPCR)

BIT	0	1 2 3	4	5	6	7	8	9	10	11	12 13	14	15
FIELD	RST	USBCMD	1	1	1	1	0	0	0	0	ENDPOINT	RES	FLG
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	RW/	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x9C0												

- RST—The reset bit is set by the core and cleared by the communication processor module and when this command is executed, RST and FLG are cleared within two general system clocks. The RISC microcontroller reset routine is approximately 60 clocks long, but you can start initializing the communication processor module immediately after this command is issued. RST is useful when the core wants to reset the registers and parameters for all the channels as well as the RISC microprocessor and timer tables. However, this bit does not affect the serial interface or parallel I/O registers.
- USBCMD—This field contains the USB command
  - 001 = The **STOP TX ENDPOINT** command disable the transmission of data on the selected endpoint. After you issue the command, flush the corresponding endpoint FIFO. No further transmissions will occur until the **RESTART TX ENDPOINT** command is issued.
  - 010 = The **RESTART TX ENDPOINT** command enables the transmission of data from the corresponding endpoint on the USB. This command is expected by the USB controller after a **STOP TX ENDPOINT** command or after a transmission error (underrun or timeout) occurs.
  - 000 = Reserved.
  - 011 = Reserved.
  - 100 = Reserved.
  - 101 = Reserved.
  - 110 = Reserved.
  - 111 = Reserved.
- ENDPOINT—This bit is the logical pipe number.
  - 00 = Endpoint 0.
  - 01 = Endpoint 1.
  - 10 = Endpoint 2.
  - 11 = Endpoint 3.

- FLG—The bit is set by the core and cleared by the communication processor module.
  - 0 = The communication processor module is ready to receive a new command.
  - 1 = The CPCR contains a command that the communication processor module is currently processing. The communication processor module clears this bit when the command finishes executing or after reset.

For additional opcode information, see Table 16-2.

16.10.7 USB Controller Errors

The USB controller reports frame reception and transmission error conditions using the channel buffer descriptors and the USB event register. The following transmission errors can be detected by the USB controller.

- Transmit Underrun Error—If this error occurs, the channel forces a bit-stuffing violation, terminates buffer transmission, closes the buffer, sets the UN bit in the TX buffer descriptor, and sets the corresponding TXE bit in the USB event register. The endpoint resumes transmission after the **RESTART TX ENDPOINT** command is received.
- Transmit Timeout Error—If this error occurs, the channel tries to retransmit if the RTE bit is set in the USB endpoint configuration register. If the RTE bit is not set or the second attempt fails, the channel closes the buffer, sets the TO bit in the TX buffer descriptor, and sets the corresponding TXEx bit in the USB event register. The endpoint resumes transmission after the **RESTART TX ENDPOINT** command is received.
- TX Data Not Ready Error—This error occurs if an IN token was received, but the corresponding endpoint's FIFO was empty, or if the endpoint was configured to NAK or STALL. The channel will set the TXEx bit in the USB event register.

The following reception errors can be detected by the USB controller.

- Overrun Error—The USB controller maintains an internal FIFO for receiving data. If a receive overrun occurs, the channel writes the received data byte to the internal FIFO over the previously received byte. The channel closes the buffer, sets the OV bit in the buffer descriptor, and sets the RXB bit in the USB event register. The NAK handshake is transmitted at the end of the received packet if the packet was error-free.
- Busy Error—A frame was received and discarded due to a lack of buffers. The channel sets the BSY bit in the USB event register.
- Non Octet Aligned Packet Error—If this error occurs, the channel writes the received data to the data buffer, closes the buffer, sets the NO bit in the RX buffer descriptor, and generates a RXB interrupt.
- CRC Error—When a CRC error occurs, the channel closes the buffer, and sets the CR bit in the RX buffer descriptor and the RXB bit in the USB event register. In isochronous mode, the USB controller reports a CRC error, however, there are no handshake packets (ACK) and the transfer continues normally when an error occurs.

16.10.8 USB Controller Programming Model

16.10.8.1 USB MODE REGISTER. The read/write USB mode (USMOD) register controls the USB controller's operation mode.

USMOD

BIT	0	1	2	3	4	5	6	7
FIELD	LSS	RESUME	RESERVED			TEST	HOST	EN
RESET	0	0	0			0	0	0
R/W	R/W	R/W	R/W			R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA00							

LSS—Low-Speed Signaling

When set, this bit selects low-speed (1.5Mbps) signaling. The actual bit rate depends on the USB clock source.

0 = Full-speed (12Mbps) signaling. Normal operation mode.
1 = Low-speed (1.5Mbps) signaling. This mode can be used for a point-to-point connection to a low-speed device or in local loopback mode.

RESUME—Generate Resume Condition

When set, this bit generates a resume condition on the USB. This bit should be used if the function wants to exit suspend mode.

Bits 2–4—Reserved

These bits are reserved and should be set to zero.

TEST—Test Mode

0 = Normal operation.
1 = Local loopback mode. In this mode, if the HOST bit is set, endpoint 1 operates as host and endpoints 1–3 can be used as function endpoints.

HOST—Host Mode

0 = The USB controller implements a USB function.
1 = The USB controller implements a USB host.

EN—Enable USB

This bit enables USB operation. When the EN bit is cleared, the USB is in a reset state and consumes minimal power.

0 = USB is disabled.
1 = USB is enabled.

Note: You should not modify any other bits of the USMOD register while the EN bit is set.

16.10.8.2 USB RECEIVE BUFFER DESCRIPTOR. The USB controller reports information about each buffer of received data using (RX) buffer descriptors. The USB controller closes the current buffer, generates a maskable interrupt, and starts receiving data in the next buffer when the current buffer is full. Additionally, it will close the buffer under the following conditions:

- When an end of packet is detected
- When an overrun error occurs
- When a bit stuff violation is detected

Note: The buffer descriptor's status bits are sticky. Applications that operate on those bits may need to be reset after every relevant CPM transaction.

The first word of the RX buffer descriptor contains status and control bits. You should prepare these bits before reception because they are set by the USB controller after the buffer has been closed. The second word contains the data length, in bytes, that was received. The third and fourth words contain a pointer that always points to the beginning of the received data buffer.

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	**E**	RES	**W**	**I**	L	F	RES		PID		RES	NO	AB	CR	OV	RES
OFFSET + 2	DATA LENGTH															
OFFSET + 4	**TX DATA BUFFER POINTER**															
OFFSET + 6																

NOTE: You are only responsible for initializing the items in bold.

E—Empty

0 = The data buffer associated with this RX buffer descriptor has been filled with received data or data reception has been aborted due to an error condition. The core is free to examine or write to any fields of this RX buffer descriptor. The communication processor module will not use this buffer descriptor again when the E bit is zero.

1 = The data buffer associated with this buffer descriptor is empty or reception is currently in progress. This RX buffer descriptor and its associated receive buffer are owned by the communication processor module. Once the E bit is set, the core should not write any fields to this RX buffer descriptor.

Bits 1, 9–10, and 15—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the RX buffer descriptor table.

1 = This is the last buffer descriptor in the RX buffer descriptor table. After this buffer is used, the communication processor module receives incoming data into the first buffer descriptor in the table (the buffer descriptor pointed to by RBASE). The number of RX buffer descriptors in this table is programmable and determined only by the W bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer has been filled.

1 = The RXB bit in the USB event register will be set when this buffer has been completely filled by the communication processor module, thus indicating the need for the core to process the buffer. The RXB bit can cause an interrupt if it is enabled. This bit is written by the USB controller after the received data has been placed into the associated data buffer.

L—Last

This bit is set by the USB controller when the buffer is closed due to detection of end-of-packet condition on the bus or as a result of an error. This bit is written by the USB controller after the received data has been placed into the associated data buffer.

0 = This buffer does not contain the last character of the message.

1 = This buffer contains the last character of the message.

F—First

This bit is set by the USB controller when the buffer contains the first byte of a packet. This bit is written by the USB controller after the received data has been placed into the associated data buffer.

0 = This buffer does not contain the first byte of the message.

1 = This buffer contains the last byte of the message.

PID—Packet ID

This field is set by the USB controller to indicate the type of the packet. It is only valid if the F bit is set. This field is written by the USB controller after the received data has been placed into the associated data buffer.

00 = This buffer contains a DATA0 packet.
01 = This buffer contains a DATA1 packet.
1X = Reserved.

NO—RX Non Octet Aligned Packet

This bit indicates that a packet containing a number of bits not exactly divisible by eight has been received. This bit is written by the USB controller after the received data has been placed into the associated data buffer.

AB—Frame Aborted

This bit indicates that a bit stuff error has occurred during reception. This bit is written by the USB controller after the received data has been placed into the associated data buffer.

CR—CRC Error

This bit indicates that a frame contains a CRC error. The received CRC bytes are always written to the receive buffer. This bit is written by the USB controller after the received data has been placed into the associated data buffer.

OV—Overrun

This bit indicates that a receiver overrun has occurred during reception. This bit is written by the USB controller after the received data has been placed into the associated data buffer.

DATA LENGTH

This field represents the number of octets that the communication processor module has written into this buffer descriptor's data buffer. The communication processor module writes to this field when the buffer descriptor is closed. This bit is written by the USB controller after the received data has been placed into the associated data buffer.

Note: The actual amount of memory allocated for this buffer should be equal to the contents of the MRBLR, plus two CRC bytes that are included in the RX buffer. The USB device driver may strip out these two bytes before the data is sent to your application.

RX DATA BUFFER POINTER

This field always points to the first location of the associated data buffer and must be divisible by four. The buffer may reside in either internal or external memory. This bit is written by the USB controller after the received data has been placed into the associated data buffer.

16.10.8.3 USB TRANSMIT BUFFER DESCRIPTOR. Data to be transmitted with the USB is presented to the communication processor module by arranging it in buffers referenced by the transmit (TX) buffer descriptor ring. The first word of the TX buffer descriptor contains status and control bits.

BIT	0	1	2	3	4	5	6	7	8 9	10	11	12	13	14	15
OFFSET + 0	R	RES	W	I	L	TC	CNF	RES	PID	RES	NAK	STAL	TO	UN	RES
OFFSET + 2	DATA LENGTH														
OFFSET + 4	TX DATA BUFFER POINTER														
OFFSET + 6															

NOTE: You are only responsible for initializing the items in bold.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

R—Ready

0 = The data buffer associated with this buffer descriptor is not ready for transmission. You are free to manipulate this buffer descriptor or its associated data buffer. The communication processor module clears this bit after the buffer has been transmitted or after an error condition is encountered.

1 = The data buffer, which you have prepared for transmission, has not been transmitted or is currently being transmitted. You cannot write to the fields of this buffer descriptor once this bit is set.

Bits 1, 6–12, and 15—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the TX buffer descriptor table.

1 = This is the last buffer descriptor in the TX buffer descriptor table. After this buffer has been used, the communication processor module will transmit incoming data from the first buffer descriptor in the table (the buffer descriptor pointed to by TBASEx). The number of TX buffer descriptors in this table is programmable and is determined only by the W bit and the overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer has been serviced.
1 = The TXB or the appropriate TXEx bit in the USBER is set when this buffer is serviced. TXB and TXEx can cause interrupts if they are enabled.

L—Last

0 = This buffer does not contain the last character of the message.
1 = This buffer contains the last character of the message.

TC—Transmit CRC

This bit is only valid when the L bit is set. Otherwise, it is ignored.

0 = Transmit end-of-packet after the last data byte. This setting can be used for testing purposes to send a bad CRC after the data.
1 = Transmit the CRC sequence after the last data byte.

CNF—Transmit Confirmation

This bit is only valid when the L bit is set. Otherwise, it is ignored.

0 = Continue to load the transmitter FIFO with the next packet. No handshake or response is expected from the function for this packet.
1 = Wait for handshake or response from the function before stating the next packet or this is the last packet.

Note: You cannot set CNF = 0 for a token that should be followed by a data packet if the data packet buffer descriptor is not ready.

PID—Packet ID

This field is set by the USB controller to indicate the type of the packet. It is only valid if the F bit is set. This field is written by the USB controller after the received data has been placed into the associated data buffer.

NAK—NAK Handshake Received

This bit indicates that the endpoint has responded with a NAK handshake. The packet was received error-free, however, the endpoint could not accept it. This bit is written by the USB controller after it has finished transmitting the associated data buffer.

STAL—STALL Handshake Received

This bit indicates that the endpoint has responded with a STALL handshake. The endpoint needs attention through the control pipe. This bit is written by the USB controller after it has finished transmitting the associated data buffer.

TO—Time-Out

This bit indicates that the endpoint has failed to acknowledge this packet. This bit is written by the USB controller after it has finished transmitting the associated data buffer.

UN—Underrun

This bit indicates that the USB has encountered a transmitter underrun condition while transmitting the associated data buffer. This bit is written by the USB controller after it has finished transmitting the associated data buffer.

DATA LENGTH

This field represents the number of octets that the communication processor module should transmit from this buffer descriptor's data buffer. It is never modified by the communication processor module. This value should normally be greater than zero. This bit is written by the USB controller after it has finished transmitting the associated data buffer.

TX DATA BUFFER POINTER

This field always points to the first location of the associated data buffer and it can be even or odd.The buffer can reside in either internal or external memory. This bit is written by the USB controller after it has finished transmitting the associated data buffer.

16.10.8.4 USB SLAVE ADDRESS REGISTER. The 8-bit, memory-mapped, read/write USB address register (USADR) holds the address for this USB port in slave mode.

USADR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED	SAD						
RESET	0	0						
R/W	R/W	R/W						
ADDR	(IMMR & 0xFFFF0000) + 0xA01							

Bit 0—Reserved

This bit is reserved and should be set to 0.

SAD—Slave Address 0–6

This field contains the slave address for the USB port.

16.10.8.5 USB COMMAND REGISTER. The 8-bit, read/write USB command (USCOM) register is used to start USB transmit operation in slave mode.

USCOM

BIT	0	1	2	3	4	5	6	7
FIELD	STR	FLUSH	RESERVED				EP	
RESET	0	0	0				0	
R/W	R/W	R/W	R/W				R/W	
ADDR	(IMMR & 0xFFFF0000) + 0xA02							

STR—Start FIFO Fill

When set, this bit causes the USB controller to start filling the corresponding endpoint transmit FIFO with data. Actual transmission will begin once the IN token for this endpoint is received. The STR bit is read always as a zero.

FLUSH—Flush FIFO

When set, this bit causes the USB controller to flush the corresponding endpoint transmit FIFO. Before flushing the FIFO, you should issue the **STOP TX ENDPOINT** command. After flushing the FIFO you should issue the **RESTART TX ENDPOINT** command (see **Section 16.10.6 USB Commands** for more information). The FLUSH bit is always read as a zero.

Bits 2–5—Reserved

These bits are reserved and should be set to zero.

EP—Endpoint

This field selects one of the four supported endpoints.

16.10.8.6 USB ENDPOINT REGISTERS 0–3. There are four 16-bit, memory-mapped, read/write USB endpoint configuration (USEPx) registers.

USEP0–USEP3

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	EPN				RESERVED		TM		RESERVED		MF	RTE	THS		RHS	
RESET	0				0		0		0		0	0	0			
R/W	R/W				R/W		R/W		R/W		R/W	R/W	R/W		R/W	
ADDR	(IMMR & 0xFFFF0000) + 0xA04 (USEP0), 0xA06 (USEP1), 0xA08 (USEP2), 0xA0A (USEP3)															

EPN— Endpoint Number

This field defines the supported endpoint number. This field is only used in slave mode and is ignored in host mode.

TM—Transfer Mode

00 = Control.
01 = Interrupt.
10 = Bulk.
11 = Isochronous.

MF—Enable Multi-Frame

This bit allows loading of the next transmit packet into the FIFO before the previous packet finishes transmitting. This bit should be set to zero unless the endpoint is configured for ISO transfer mode.

0 = The transmit FIFO can hold only one packet.
1 = The transmit FIFO can hold more than one packet.

RTE—Retransmit enable

This bit should be set to zero for an endpoint configured for ISO transfer mode.

0 = No retransmission.
1 = Automatic frame retransmission is enabled. The frame is retransmitted if transmit an error occurred (time-out).

Note: The RTE bit can only be set if the transmit packet is contained in a single buffer. Other wise, retransmission should be handled by software intervention.

THS—Transmit Handshake (Slave Mode Only)

00 = Normal handshake.
01 = Ignore IN token.
10 = Force NAK handshake. Not allowed for control endpoint.
11 = Force STALL handshake. Not allowed for control endpoint.

RHS—Receive Handshake (Slave Mode Only)

00 = Normal handshake.
01 = Ignore OUT token.
10 = Force NAK handshake. Not allowed for control endpoint.
11 = Force STALL handshake. Not allowed for control endpoint.

16.10.8.7 USB BUFFER DESCRIPTOR RING. The data associated with the USB channel is stored in buffers that are referenced by buffer descriptors organized in buffer descriptor rings located in the dual-port RAM. These rings have the same basic configuration as those used by the serial communication controller and serial management controllers. There are four separate transmit buffer descriptor rings and four separate receive buffer descriptor rings for each endpoint, as illustrated in Figure 16-112.

The buffer descriptor ring allows you to define buffers for transmission and buffers for reception. Each buffer descriptor ring forms a circular queue. The communication processor module confirms reception and transmission or indicates error conditions using the buffer descriptors to inform the processor that the buffers have been serviced. The actual buffers can reside in either external memory or internal memory. Data buffers can reside in the parameter RAM of a serial communication controller if it is not enabled.

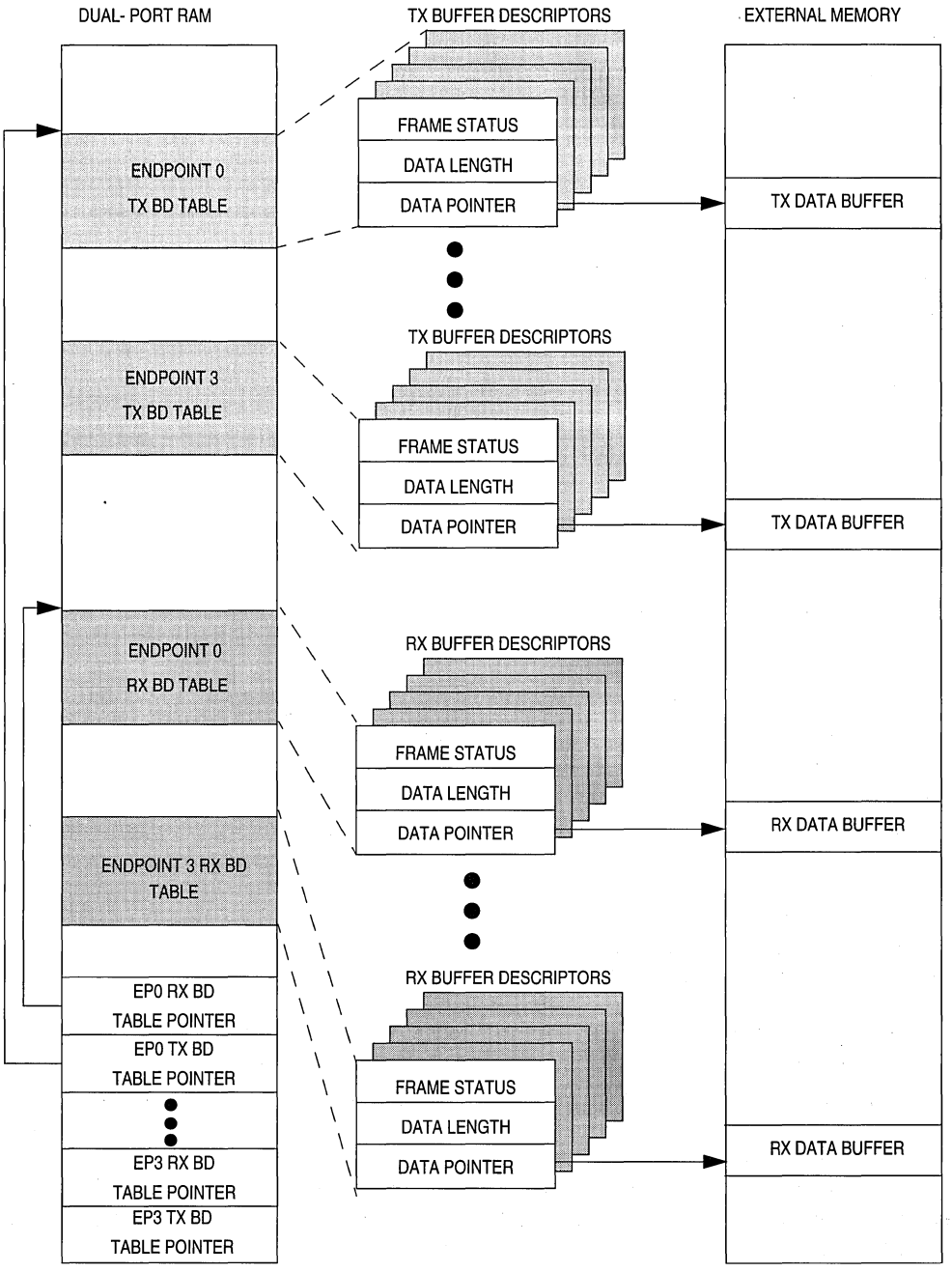


Figure 16-112. USB Buffer Descriptor Ring

16.10.8.8 USB EVENT REGISTER. The 16-bit, memory-mapped USB event register (USBER) is used to generate interrupt events and report events recognized by the USB channel. When an event is recognized, the USB sets the corresponding bit in the USBER. Interrupts generated by this register may be masked in the USBMR. A bit is cleared by writing a one (writing a zero has no effect) and more than one bit can be cleared at a time. All unmasked bits must be cleared before the communication processor module will clear the internal interrupt request.

USBER

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED						RESET	IDLE	TXE3	TXE2	TXE1	TXE0	SOF	BSY	TXB	RXB
RESET	0						0	0	0	0	0	0	0	0	0	0
R/W	R/W						R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0XFFFF0000) + 0XA10															

Bits 0–5—Reserved

These bits are reserved and should be set to 0.

RESET—Reset Condition Detected

This bit indicates that the USB reset condition has been asserted.

IDLE—IDLE Status Changed

This bit indicates that a change in the status of the serial line was detected. The real time suspend status is reflected in the USBS register.

TXEx—TX Error

This bit indicates that an error occurred during transmission for endpoint x (packet not acknowledged or underrun).

SOF—Start of Frame

This bit indicates that a start-of-frame packet was received. The packet is stored in the FRAME_N parameter RAM entry

BSY—Busy Condition

This bit indicates that received data has been discarded due to a lack of buffers. This bit is set after the first character is received for which there is no receive buffer available.

TXB—TX Buffer

This bit indicates that a buffer has been transmitted. It is set once the transmit data of the last character in the buffer was written to the transmit FIFO (if L=0) or after the last character was transmitted on the line (if L=1).

RXB—RX Buffer

This bit indicates that a buffer has been received. It is set after the last character has been written to the receive buffer and the RX buffer descriptor is closed.

16.10.8.9 USB MASK REGISTER. The 16-bit read/write USB mask register (USBMR) has the same bit formats as the USB event register. If a bit in the USBMR is one, the corresponding interrupt in the USBER is enabled. If the bit is zero, the corresponding interrupt in the USBER will be masked.

USBMR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED						RESET	IDLE	TXE3	TXE2	TXE1	TXE0	SOF	BSY	TXB	RXB
RESET	0						0	0	0	0	0	0	0	0	0	0
R/W	R/W						R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0XFFFF0000) + 0XA14															

16.10.8.10 USB STATUS REGISTER. The 8-bit read-only USB status (USBS) register allows you to monitor real-time status condition on the USB lines.

USBS

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED							IDLE
RESET	0							0
R/W	R/W							R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA17							

Bits 0–6—Reserved

These bits are reserved and should be set to 0.

IDLE—Idle Status

When set, this bit indicates that an idle condition has been detected on the USB lines. It is cleared when the bus is not idle.

16.10.8.11 USB CONTROLLER INITIALIZATION EXAMPLE. The following is an example initialization sequence for the USB controller. It can be used to set up four function endpoints (0–3) to fill up transmit FIFOs so that data is ready for transmission when an IN token is received from the USB. You can generate the token by using a USB traffic generator.

1. Write 0x00010000 to the BRGC1 register for division factor 1 to produce 48MHz, assuming the system clock is 48MHz.
2. Clear the DR14 and DR15 bits of the PADIR and the DD14 and DD15 bits of the PAPAR to select the USBRXD and $\overline{\text{USBOE}}$ pins.
3. Clear the DR10 and DR11 bits of the PCDIR, the DD10 and DD11 bits of the PCPAR, and set the CD1 and CTS1 bits of the PCSO to select the USBRXP and USBRXN pins.

4. Set the DR6 and DR7 bits of the PCDIR and the DD6 and DD7 bits of the PCPAR to select the USBTXP and USBTXN pins.
5. Write DPRAM+500 to the EP0PTR and DPRAM+520 to the EP1PTR to set up the endpoint 0 and 1 pointers.
6. Write DPRAM+540 to the EP2PTR and DPRAM+560 to the EP3PTR to set up the endpoint 2 and 3 pointers.
7. Write 0xBC800004 to DPRAM+20 to set up the various status and DATA LENGTH fields of the endpoint 0 TX buffer descriptor.
8. Write DPRAM+200 to DPRAM+24 to set up the TX DATA BUFFER POINTER field of the endpoint 0 TX buffer descriptor.
9. Write BCC00004 to DPRAM +28 to set up the status and DATA LENGTH fields of the endpoint 1 TX buffer descriptor.
10. Write DPRAM+210 to DPRAM+2C to set up the DATA BUFFER POINTER field of the endpoint 1 TX buffer descriptor.
11. Write BC80004 to DPRAM+30 to set up the status and length fields of the endpoint 2 TX buffer descriptor.
12. Write DPRAM +220 to DPRAM+34 to set up the DATA BUFFER POINTER field of the endpoint 2 TX buffer descriptor.
13. Write BCC00004 to DPRAM+30 to set up the status and length fields of the endpoint 3 TX buffer descriptor.
14. Write DPRAM+30 to DPRAM+3C to set up the DATA BUFFER POINTER field of the endpoint 3 TX buffer descriptor.
15. Write CAFECAFE to DPRAM+200 to set up the endpoint 0 TX data pattern.
16. Write FACEFACE to DPRAM+210 to set up the endpoint 1 TX data pattern.
17. Write BACEBACE to DPRAM+220 to set up the endpoint 2 TX data pattern.
18. Write CaCeCaCe to DPRAM+230 to set up the endpoint 3 TX data pattern.
19. Write 20002020 to DPRAM+500 to set up the RBASE and TBASE fields of the endpoint 0 parameter RAM.
20. Write 18180100 to DPRAM+504 to set up the RFCR and TFCR fields of the endpoint 0 parameter RAM.
21. Write 20002020 to DPRAM+508 to set up the RBPTR and TBPTR fields of the endpoint 0 parameter RAM.
22. Clear the TSTATE field of the endpoint 0 parameter RAM.
23. Write 20082028 to DPRAM+520 to set up the RBASE and TBASE fields of the endpoint 1 parameter RAM.
24. Write 18180100 to DPRAM+524 to set up the RFCR and TFCR fields of the endpoint 1 parameter RAM.
25. Write 20082028 to DPRAM+528 to set up the RBPTR and TBPTR fields of the endpoint 1 parameter RAM.
26. Clear the TSTATE field of the endpoint 1 parameter RAM.

27. Write 20102030 to DPRAM+540 to set up the RBASE and TBASE fields of the endpoint 2 parameter RAM.
28. Write 18180100 to DPRAM+544 to set up the RFCR and TFCR fields of the endpoint 2 parameter RAM.
29. Write 20102030 to DPRAM+548 to set up the RBPTR and TBPTR fields of the endpoint 2 parameter RAM.
30. Clear the TSTATE field of the endpoint 2 parameter RAM.
31. Write 20182038 to DPRAM+560 to set up the RBASE and TBASE fields of the endpoint 3 parameter RAM.
32. Write 18180100 to DPRAM+564 to set up the RFCR and TFCR fields of the endpoint 3 parameter RAM.
33. Write 20182038 to DPRAM+568 to set up the RBPTR and TBPTR fields of the endpoint 3 parameter RAM.
34. Clear the TSTATE field of the endpoint 3 parameter RAM.
35. Write 0x0200 to the USEP0 for endpoint 0, bulk transfer, one packet only, and manual handshake.
36. Write 0x1200 to the USEP1 for endpoint 1, bulk transfer, one packet only, and manual handshake.
37. Write 0x2200 to the USEP2 for endpoint 2, bulk transfer, one packet only, and manual handshake.
38. Write 0x3200 to the USEP3 for endpoint 3, bulk transfer, one packet only, and manual handshake.
39. Write 0x00 to the USMOD for full-speed 12mps function endpoint mode and disable the USB.
40. Write 0x05 to the USAD for slave address 5.
41. Clear the USCOM register.
42. Set the EN bit in the USMOD register to enable the USB controller.
43. Write 0x80 to the USCOM to start filling the TX FIFO with endpoint 0 data ready for transmission when IN token is received.
44. Write 0x81 to the USCOM to start filling the TX FIFO with endpoint 1 data ready for transmission when IN token is received.
45. Write 0x82 to the USCOM to start filling the TX FIFO with endpoint 2 data ready for transmission when IN token is received.
46. Write 0x83 to the USCOM to start filling the TX FIFO with endpoint 2 data ready for transmission when IN token is received.

16.11 THE SERIAL MANAGEMENT CONTROLLERS

The serial management controllers (SMCs) consist of two full-duplex ports that can be independently configured to support any one of three protocols or modes—UART, Transparent, or general-circuit interface (GCI). Simple UART operation is used to provide a debug/monitor port in an application, which allows the serial communication controller to be free for other purposes. The serial management controller clock can be derived from one of the four internal baud rate generators or from a 16× external clock pin.

The functionality and performance provided by the SMC in UART mode is generally less than that provided by the SCC2 in UART mode. The SMC implementation does not support such features as special character recognition and detection.

In totally Transparent mode, the serial management controller can use the TDM channel to connect to a T1 line or directly to the SMC's set of pins. However, SMC2 does not have its own set of dedicated pins, so the time-slot assigner pins are its only option. The receive and transmit clocks are derived from the time-division multiplex (TDM) channel, the internal baud rate generators, or from an external 1× clock. The Transparent protocol allows the transmitter and receiver to use the external synchronization pin.

Each serial management controller supports the circuit interface and monitor channels of the GCI bus. In which case, the serial management controller is connected to a TDM channel in the serial interface. For testing purposes, the serial management controllers support loopback and echo modes. The SMC receiver and transmitter are double-buffered, which provides an effective FIFO size (latency) of two characters. Refer to **Section 16.7 The Serial Interface with Time-Slot Assigner** for details about configuring the GCI interfaces.

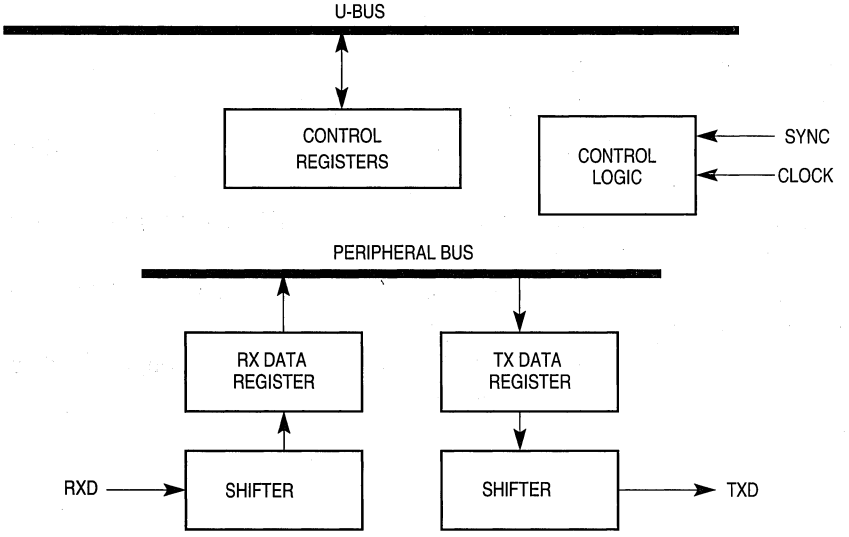


Figure 16-113. Serial Management Controller Block Diagram

The receive data source for the two serial management controller channels have different pin options for each channel. SMC1 can either use the L1RXDA pin of the serial interface or the SMRXD1 pin if it is connected to the NMSI. SMC2 can also use the L1RXDA pin of the serial interface, but if you use the SMRXD2 pin the serial interface time-slot assigner function is not available. Likewise, the transmit data source for SMC1 can be the L1TXDA pin if the serial management controller is connected to the TDM or the SMTXD1 pin if it is connected to the NMSI. SMC2 transmit data source can also be L1TXDA pin if the serial management controller is connected to the TDM, but if you use the SMTXD2 pin the serial interface time-slot assigner function is not available.

If the serial management controllers are connected to the TDM, the SMC receive and transmit clocks can be independent from each other, as defined in the CRTA bit of the SIMODE register. Refer to **Section 16.7.5.2 Serial Interface Mode Register** for more information. However, if the serial management controller is connected to the NMSI, the SMC receive and transmit clocks must be connected to a single clock source called SMCLK, which is an internal signal name for a clock that is generated from the bank of clocks. SMCLK originates from an external pin or one of the two internal baud rate generators. Refer to **Section 16.7.8 Nonmultiplexed Serial Interface Configuration** for more details.

If the serial management controllers are connected to the TDM, it gets its synchronization pulse from the time-slot assigner. Otherwise, if the serial management controller is connected to the NMSI and the Transparent protocol is selected, the serial management controller can use the $\overline{\text{SMSYNx}}$ pin as a synchronization pin to determine when it should start transmitting and receiving. The $\overline{\text{SMSYNx}}$ pin is not used when the serial communication controller is in UART mode.

16.11.1 Features

The following is a list of the serial management controllers' main features:

- Each serial management controller can implement the UART protocol on its own pins
- Each serial management controller can implement a totally Transparent protocol on a multiplexed or nonmultiplexed line. If SMC2 uses a nonmultiplexed line, the serial interface time-slot assigner is not available.
- Each SMC channel fully supports the circuit interface and monitor channels of the GCI (IOM-2) in ISDN applications
- Two serial management controllers support the two sets of circuit interface and monitor channels in the SCIT channels 0 and 1
- Full-duplex operation
- Local loopback and echo capability for testing

16.11.2 General SMC Mode Register

The operating mode of each serial management controller port is defined by the 16-bit, memory-mapped, read/write general SMC mode register (SMCMR). Refer to each specific SMC protocol section for more information on this register and Table 16-2 for specific commands.

16.11.3 SMC Buffer Descriptor Operation

When the serial management controllers are configured to operate in GCI mode, their memory structure is predefined to be one half-word long for transmit and one half-word long for receive. For more information on these half-word structures, refer to **Section 16.11.8 The SMC in GCI Mode**.

In UART and transparent modes, the serial management controllers have a memory structure that is like that of the serial communication controller. Each buffer is referenced by a buffer descriptor and organized in a buffer descriptor ring located in the dual-port RAM. Refer to Figure 16-114 for details.

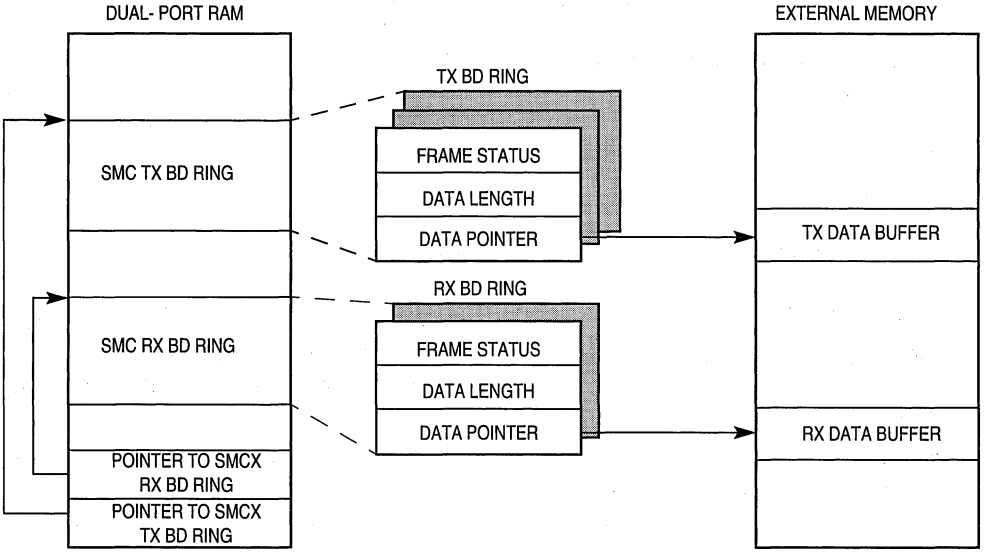


Figure 16-114. SMC Memory Format

The buffer descriptor ring allows you to define buffers for transmission and reception and each ring forms a circular queue. Using the buffer descriptors, the communication processor module confirms reception and transmission so that the processor knows the buffers have been serviced. The data buffers can reside in external or internal memory and the data buffers reside in the parameter area of an SCC2 or SMC if that channel is not enabled.

16.11.4 SMC General Parameter RAM Memory Map

Each SMC parameter RAM area begins at the same offset from each base. The protocol-specific portions of the SMC parameter RAM are discussed in each mode. The SMC general parameter RAM shared by the UART and transparent protocols is shown in Table 16-36. The GCI protocol has its own parameter RAM.

You must initialize certain parameter RAM values before the serial management controller is enabled. Other values are initialized or written by the communication processor module. Once initialized, most parameter RAM values do not need to be accessed in your software since most of the activity is centered around the transmit and receive buffer descriptors and not the parameter RAM. However, if you access the parameter RAM, note the following restrictions.

- The parameter RAM values that pertain to the SMC transmitter can only be written when the TEN bit in the SMCMR is zero. Or, after the **STOP TRANSMIT** command is issued, but before the **RESTART TRANSMIT** command is issued.
- The parameter RAM values that pertain to the SMC receiver can only be written when:
  - ❑ The REN bit in the SMCMR is zero.
  - ❑ The receiver is previously enabled after the **ENTER HUNT MODE** command is issued.
  - ❑ The **CLOSE RX BD** command is issued and before the REN bit is set.

Table 16-36. SMC (UART and Transparent) Parameter RAM Memory Map

ADDRESS	NAME	WIDTH	DESCRIPTION
SMC Base + 00	**RBASE**	Half-Word	RX BD Base Address
SMC Base + 02	**TBASE**	Half-Word	TX BD Base Address
SMC Base + 04	**RFCR**	Byte	RX Function Code
SMC Base + 05	**TFCR**	Byte	TX Function Code
SMC Base + 06	**MRBLR**	Half-Word	Maximum Receive Buffer Length
SMC Base + 08	RSTATE	Word	RX Internal State
SMC Base + 0C	RPTR	Word	RX Internal Data Pointer
SMC Base + 10	RBPTR	Half-Word	RX Buffer Descriptor Pointer
SMC Base + 12	RCNT	Half-Word	RX Internal Byte Count
SMC Base + 14	RTMP	Word	RX Temp
SMC Base + 18	TSTATE	Word	TX Internal State
SMC Base + 1C	TPTR	Word	TX Internal Data Pointer
SMC Base + 20	TBPTR	Half-Word	TX Buffer Descriptor Pointer
SMC Base + 22	TCNT	Half-Word	TX Internal Byte Count
SMC Base + 24	TTMP	Word	TX Temp
SMC Base + 28			First Word of Protocol-Specific Area
SMC Base + 36			Last Word of Protocol-Specific Area

NOTE: You are only responsible for initializing the items in bold.
SMC Base = (IMMR & 0xFFFF0000) + 0x3E80 (SMC1) and 0x3F80 (SMC2).

- RBASE and TBASE—These entries are where the dual-port RAM starts the SMC receive and transmit buffer descriptors. They provide a great deal of flexibility in how buffer descriptors for a serial management controller are partitioned. By selecting RBASE and TBASE entries for the serial management controller and by setting the W bit in the last buffer descriptor in each buffer descriptor list, you can select how many buffer descriptors to allocate for the transmit and receive side of the serial management controller. However, you must initialize these entries before enabling the corresponding channel. Furthermore, you should not configure buffer descriptor tables of two enabled serial management controllers to overlap or erratic operation will occur.

Note: RBASE and TBASE should contain a value that is divisible by eight.

- RFCR and TFCR—There are separate function code registers for the two SMC channels. One for receive data buffers (RFCR) and one for transmit data buffers (TFCR). The function code entry contains the value that you want to appear on the AT pins when the associated SDMA channel accesses memory. It also controls the byte-ordering convention that is used in the transfers.

RFCR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED			BO		AT		
RESET	0			0		0		
R/W	R/W			R/W		R/W		
ADDR	SMC1 AND SMC2 BASE + 0x04							

Bits 0–2—Reserved

These bits are reserved and should be set to 0.

BO—Byte Ordering

You should set these bits to select the required byte ordering of the data buffer. If this field is modified on-the-fly, it takes effect at the beginning of the next frame or the next buffer descriptor.

00 = The DEC/Intel convention is used for byte ordering (swapped operation) and is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation) is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT—Address Type

These bits contain the function code value used during the SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA type.

TFCR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED			BO		AT1–AT3		
RESET	0			0		0		
R/W	R/W			R/W		R/W		
ADDR	SMC1 AND SMC2 BASE + 0x05							

Bits 0–2—Reserved

These bits are reserved and should be set to 0.

BO—Byte Ordering

You should set these bits to select the required byte ordering of the data buffer. If this field is modified on-the-fly, it takes effect at the beginning of the next frame or the next buffer descriptor.

00 = The DEC/Intel convention is used for byte ordering (swapped operation) and is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation) is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT—Address Type

These bits contain the function code value used during this SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA type.

- MRBLR—Each serial management controller has one maximum receive buffer length register to define the receive buffer length. The MRBLR defines the maximum number of bytes that the MPC823 writes to a receive buffer on the serial management controller before it moves on to the next buffer. The MPC823 can write fewer bytes to the buffer than MRBLR if a condition, such as an error or end-of-frame occurs, but it never writes more bytes than the MRBLR value. It follows then, that buffers you supply should always be at least as long as the MRBLR. The transmit buffers for a serial management controller are not affected in any way by the value programmed into the MRBLR. Transmit buffers can be individually chosen to have varying lengths. The number of bytes to be transmitted is chosen by programming the DATA LENGTH field in the TX buffer descriptor.

Note: The MRBLR is not intended to be changed dynamically while a serial management controller is operating. However, if it is modified in a single bus cycle with one 16-bit move (not two 8-bit bus cycles back-to-back) then a dynamic change in receive buffer length can be successfully achieved. This occurs when the communication processor module moves control to the next RX buffer descriptor in the table. Thus, a change to MRBLR does not have an immediate effect. To guarantee the exact RX buffer descriptor on which the change occurs, you should only change the MRBLR while the SMC receiver is disabled. The value of MRBLR should be greater than zero and should be even if the character length of the data is greater than 8 bits.

- RBPTR—The receiver buffer descriptor pointer for each SMC channel points to the next buffer descriptor the receiver transfers data to when it is in idle state or to the current buffer descriptor during frame processing. After a reset or when the end of the buffer descriptor table is reached, the communication processor module initializes this pointer to the value programmed in the RBASE entry. Although you will not usually need to write the RBPTR in most applications, you can modify it when the receiver is disabled or when you are sure no receive buffer is currently being used.
- TBPTR—The transmitter buffer descriptor pointer for each SMC channel points to the next buffer descriptor the transmitter transfers data from when it is in idle state or to the current buffer descriptor during frame transmission. After a reset or when the end of the buffer descriptor table is reached, the communication processor module initializes this pointer to the value programmed in the TBASE register. Although you will not usually need to write TBPTR in most applications, you can modify it when the receiver is disabled or when you are sure no receive buffer is currently being used.

- Other General Parameters—You do not need to access these parameters during normal operation. They are only listed because they provide helpful information for experienced users and for debugging purposes. Additional parameters are listed in Table 16-33. The RX and TX internal data pointers are updated by the SDMA channels to show the next address in the buffer to be accessed. TCNT is a down-count value that is initialized with the TX buffer descriptors' DATA LENGTH field and decremented with every byte read by the SDMA channels. RCNT is a down-count value that is initialized with the MRBLR value and decremented with every byte written by the SDMA channels. RSTATE, TSTATE, RTEMP, TTEMP, and the reserved areas are only used by the RISC microcontroller.

Note: To extract data from a partially full receive buffer, use the **CLOSE RX BD** command.

16.11.5 Disabling the SMCs On-the-Fly

If you do not need a serial management controller, it can be disabled and reenabled later. In which case, you must follow a particular sequence of steps to ensure that the buffers are properly closed and that new data is transferred to or from a new buffer. Such a sequence is required if the parameters you want to change are not dynamic. If the register or bit description states that dynamic or on-the-fly changes are allowed, you do not have to follow the sequences and the register or bit can can be changed immediately.

Note: The serial management controller does not have to be fully disabled for you to modify the parameter RAM. Refer to the parameter RAM description for details on when parameter RAM values can be modified. If you want to disable the SCC2, USB, SMCs, SPI, and I^2C, use the CPM command register to reset the communication processor module with a single command.

16.11.5.1 DISABLING THE ENTIRE SMC TRANSMITTER. Follow these steps to fully enable or disable the SMC transmitter:

1. Issue the **STOP TRANSMIT** command. This command is recommended when a serial management controller is transmitting data since it stops transmission smoothly. If a serial management controller is not transmitting, then this command is not required. Furthermore, if you overwrite the TBPTR or the **INIT TX PARAMETERS** command is executed, this command is not required.
2. Clear the TEN bit in the SMCMR. This disables the SMC transmitter and puts it in a reset state.
3. Make modifications. You can make modifications to the SMC transmit parameters, including the parameter RAM. If you prefer to switch protocols or restore the SMC transmit parameters to their initial state, issue the **INIT TX PARAMETERS** command.
4. Issue the **RESTART TRANSMIT** command. You must do this if you did not issue the **INIT TX PARAMETERS** command in step 3.
5. Set the TEN bit in the SMCMR. Transmission now begins using the TX buffer descriptor that the TBPTR value pointed to as soon as the R bit is set in the TX buffer descriptor.

16.11.5.2 DISABLING PART OF THE SMC TRANSMITTER. Follow this shorter sequence if you prefer to reinitialize the transmit parameters to the state they had after reset.

1. Clear the TEN bit in the SMCMR.
2. Issue the **INIT TX PARAMETERS** command and make any additional modifications.
3. Set the TEN bit in the SMCMR.

16.11.5.3 DISABLING THE ENTIRE SMC RECEIVER. Follow these steps to fully enable or disable the receiver:

1. Clear the REN bit in the SMCMR. Reception is aborted immediately, which disables the SMC receiver and puts it in a reset state.
2. Make modifications to the SMC receive parameters, including the parameter RAM. If you prefer to switch protocols or restore the SMC receive parameters to their initial state, issue the **INIT RX PARAMETERS** command.
3. Issue the **CLOSE RX BD** command. You must do this if you did not issue the **INIT RX PARAMETERS** command in step 2.
4. Set the REN bit in the SMCMR. Reception begins immediately using the RX buffer descriptor that the RBPTR pointed to if the E bit is set in the RX buffer descriptor.

16.11.5.4 DISABLING PART OF THE SMC RECEIVER. Follow this shorter sequence to reinitialize the receive parameters to the state they had after reset.

1. Clear the REN bit in the SMCMR.
2. **INIT RX PARAMETERS** command and make any additional modifications.
3. Set the REN bit in the SMCMR.

16.11.5.5 SWITCHING PROTOCOLS. To switch the protocol that a serial management controller is executing without resetting the board or affecting any other serial management controller, use one command and follow these steps:

1. Clear the TEN and REN bits in the SMCMR.
2. Issue the **INIT TX AND RX PARAMS** command. It initializes the transmit and receive parameters. Make any additional modifications in the SMCMR
3. Set the TEN and REN bits in the SMCMR. The serial management controller is now enabled with the new protocol.

Tip: You can save power by clearing the TEN and REN bits of a serial management controller.

16.11.6 The SMC in UART Mode

Compared to the SCC2 in UART mode, the serial management controllers are designed to support simple debug/monitor ports instead of full-featured UART controllers. The following is a list of the features that the SMC in UART mode does not support.

- $\overline{\text{RTS}}$, $\overline{\text{CTS}}$, and $\overline{\text{CD}}$ pins
- Receive and transmit sections clocked at different rates
- Fractional stop bits
- Built-in multidrop modes
- Freeze mode for implementing flow control
- Isochronous operation (1× clock)
- Interrupts on special control character reception
- Ability to transmit data on demand using the TODR

However, the SMC in UART mode does provide one feature that the SCC2 in UART mode does not. It allows a data length of up to 14 bits, whereas the serial communication controller only goes up to 8 bits. The SMC in UART mode is also referred to as the SMC UART controller.

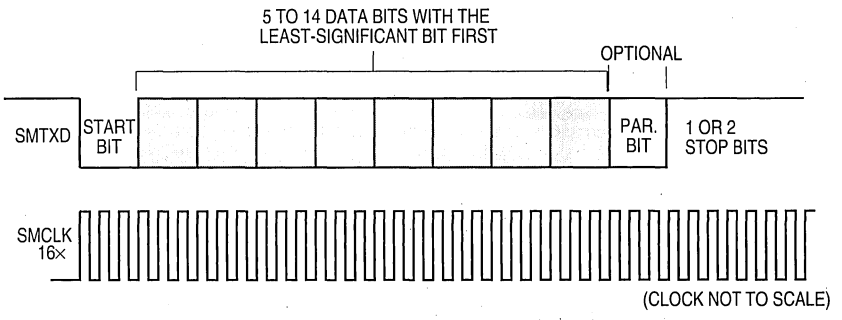


Figure 16-115. SMC UART Frame Format

16.11.6.1 FEATURES. The following list summarizes the main features of the SMC in UART mode:

- Flexible message-oriented data structure
- Programmable data length (5–14 bits)
- Programmable 1 or 2 stop bits
- Even/odd/no parity generation and checking
- Frame error, break, and idle detection
- Transmit preamble and break sequences
- Received break character length indication
- Continuous receive and transmit modes

16.11.6.2 SMC UART CHANNEL TRANSMISSION PROCESS. The UART transmitter is designed to work with almost no intervention from the core. When the core enables the SMC transmitter, it starts transmitting idles. The SMC UART controller immediately polls the first buffer descriptor in the transmit channel buffer descriptor ring and once every character time after that, depending on the character length. When there is a message to transmit, the SMC UART controller fetches the data from memory and starts transmitting the message.

When buffer descriptor data is completely written to the transmit FIFO, the SMC UART controller writes the message status bits into the buffer descriptor and clears the R bit. An interrupt is issued if the I bit in the buffer descriptor is set. If the next TX buffer descriptor is ready, the data from its data buffer is appended to the previous data and transmitted out on the transmit pin, without any gaps between the buffers. If the next TX buffer descriptor is not ready, the SMC UART controller starts transmitting idles and waits for the next TX buffer descriptor to be ready.

By appropriately setting the I bit in each buffer descriptor, interrupts can be generated after the transmission of each buffer, a specific buffer, or each block. The SMC UART controller then proceeds to the next buffer descriptor in the table. If the CM bit is set in the TX buffer descriptor, the R bit is not cleared, allowing the associated data buffer to be automatically retransmitted next time the communication processor module accesses this data buffer. For instance, if a single TX buffer descriptor is initialized with the CM and W bits set, the data buffer is continuously transmitted until you clear the R bit of the buffer descriptor.

16.11.6.3 SMC UART CHANNEL RECEPTION PROCESS. When the core enables the SMC receiver in UART mode, it enters hunt mode and waits for the first character to arrive. Once the first character arrives, the communication processor module checks the first RX buffer descriptor to see if it is empty and then starts storing characters in the associated data buffer.

When the data buffer is filled or the MAX_IDL timer expires if it is enabled, the SMC UART controller clears the E bit in the buffer descriptor and generates an interrupt if the I bit in the buffer descriptor is set. If the incoming data exceeds the length of the data buffer, the SMC UART controller fetches the next buffer descriptor in the table and, if it is empty, continues transferring data to this buffer descriptor associated data buffer. If the CM bit is set in the RX buffer descriptor, the E bit is not cleared, which allows the associated data buffer to be automatically overwritten next time the communication processor module accesses this data buffer.

16.11.6.4 SMC UART PARAMETER RAM MEMORY MAP. When configured to operate in UART mode, the SMC UART controller overlays the structure used in Table 16-36 with the parameters described in Table 16-37.

Table 16-37. SMC UART Parameter RAM Memory Map

ADDRESS	NAME	WIDTH	DESCRIPTION
SMC Base + 28	**MAX_IDL**	Half-word	Maximum Idle Characters
SMC Base + 2A	IDLC	Half-word	Temporary Idle Counter
SMC Base + 2C	BRKLN	Half-word	Last Received Break Length
SMC Base + 2E	**BRKEC**	Half-word	Receive Break Condition Counter
SMC Base +30	**BRKCR**	Half-word	Break Count Register (Transmit)
SMC Base +32	R_MASK	Half-word	Temporary Bit Mask

NOTE: You are only responsible for initializing the items in bold.
SMC Base = (IMMR & 0xFFFF0000) + 0x3E80 (SMC1) and 0x3F80 (SMC2).

- MAX_IDL—Once a character of data is received on the line, the SMC UART controller starts counting any idle characters received. If a MAX_IDL number of idle characters is received before the next data character, an idle timeout occurs and the buffer closes. This, in turn, produces an interrupt request to the core to receive the data from the buffer. MAX_IDL provides a convenient way to demarcate frames in SMC UART mode. But if you do not want to use MAX_IDL, you should program MAX_IDL to 0x0000 and the buffer will never close, regardless of the number of idle characters received. The number of bits in an idle character is calculated as follows—1 + character data length (5 to 14) + 1 (if parity bit is used) + number of stop bits (1 or 2). For example, for a character data length of 8, no parity, and 1 stop bit, the idle character length is 10 bits.
- IDLC—This value is used by the RISC microcontroller to store the current idle counter value in the MAX_IDL timeout process. IDLC is a down-counter that you do not need to initialize or access.
- BRKLN—This value is used to store the length of the last break character received and is the bit length of that character. For example, if the receive pin is low for 257 bit times, BRKLN shows the value 0x0101 and it is accurate to within one character unit of bits. For 8 data bits, no parity, 1 stop bit, and 1 start bit, BRKLN is accurate to within 10 bits.
- BRKEC—This counter counts the number of break conditions that occur on the line. One break condition can last for hundreds of bit times, yet this counter is only incremented once during that period.
- BRKCR—This value indicates when the SMC UART controller sends a break character sequence after a **STOP TRANSMIT** command is issued. The number of break characters sent by the SMC UART controller is determined by the value in BRKCR. For 8 data bits, no parity, 1 stop bit, and 1 start bit, each break character is 10 bits long and consists of all zeros.
- R_MASK—This value is a temporary bit mask used internally by the SMC UART controller.

16.11.6.5 PROGRAMMING THE SMC UART CONTROLLER. The SMC UART controller's data structure supports multibuffer operation and allows you to transmit break and preamble sequences. Overrun, parity, and framing errors are reported via the buffer descriptors. In its simplest form, the SMC UART controller functions in a character-oriented environment. Each character is transmitted with the stop bits and parity that you configure. Characters are received into separate 1-byte buffers. A maskable interrupt can be generated when each buffer is filled.

Many applications may want to take advantage of the message-oriented capabilities that the SMC UART controller supports through linked buffers for reception or transmission. You can handle data in a message-oriented environment and work on entire messages rather than on a character-by-character basis. A message can span several linked buffers and each one can be transmitted and received as a linked list of buffers without any intervention from the core, which makes it easy to program and saves processor overhead. In a message-oriented environment, the idle sequence is used as the message delimiter. The transmitter can generate an idle sequence before starting a new message and the receiver can close a buffer when an idle sequence is found.

16.11.6.6 SMC UART COMMANDS. You can program the CPM command register (CPCR) with the following commands to transmit data.

- **STOP TRANSMIT**—This command disables the transmission of characters on the transmit channel. If the SMC UART controller receives this command while transmitting a message, it stops transmitting. The SMC UART controller finishes transmitting any data that has already been transferred to its FIFO and shift register and then stops transmitting data. The TBPTR is not advanced when this command is issued. The SMC UART controller transmits a programmable number of break sequences and then transmits idles. The number of break sequences, which can be zero, should be written to the BRKCR entry before this command is issued to the SMC UART controller.
- **RESTART TRANSMIT**—This command enables characters to be transmitted on the transmit channel. The SMC UART controller expects it after disabling the channel in its SMCMR and after issuing the **STOP TRANSMIT** command. The SMC UART controller resumes transmission from the current TBPTR in the channel's TX buffer descriptor table.
- **INIT TX PARAMETERS**—This command initializes all the transmit parameters in this serial channel's parameter RAM to their reset state and should only be issued when the transmitter is disabled. The **INIT TX AND RX PARAMS** command can also be used to reset the transmit and receive parameters.

You can program the CPCR with the following commands to receive data.

- **ENTER HUNT MODE**—You should not use this command for an SMC UART channel. You should use the **CLOSE RX BD** command instead.
- **CLOSE RX BD**—This command is used to force a serial management controller to close the current receive buffer descriptor if it is currently being used and to use the next buffer descriptor in the list for any subsequently received data. If a serial management controller is not in the process of receiving data, no action is taken by this command.
- **INIT RX PARAMETERS**—This command Initializes all the receive parameters in this serial channel parameter RAM to their reset state. This command should only be issued when the receiver is disabled. The **INIT TX AND RX PARAMS** command can also be used to reset the receive and transmit parameters.

16.11.6.7 SENDING A BREAK. A break is an all-zeros character without stop bits and it is sent by issuing the **STOP TRANSMIT** command. The SMC UART controller finishes transmitting any outstanding data and then sends a character with consecutive zeros. The number of zero bits in this character is the sum of the character length, plus the number of start, parity, and stop bits. The SMC UART controller transmits a programmable number of break characters according to the BRKCR entry and then reverts to idle or sends data if the **RESTART TRANSMIT** command was issued before completion. When the break is completed, the transmitter sends at least one idle character before transmitting any data to guarantee recognition of a valid start bit.

16.11.6.8 SENDING A PREAMBLE. A preamble sequence provides a convenient way for you to ensure that the line is idle before you start a new message. The preamble sequence is constructed of consecutive ones that are one character long. If the preamble bit in a buffer descriptor is set, a serial management controller sends a preamble sequence before transmitting that data buffer. For 8 data bits, no parity, 1 stop bit, and 1 start bit, a preamble of 10 ones would be sent before the first character in the buffer. If no preamble sequence is sent, data from two ready transmit buffers can be transmitted without causing a delay on the transmit pin between the two transmit buffers.

16.11.6.9 SMC UART CONTROLLER ERRORS. The SMC UART controller reports character reception error conditions via the channel buffer descriptors and the SMC UART event register. The SMC UART controller has no transmission errors, which means you cannot stop the transmission of characters in SMC UART mode.

- Overrun Error—The SMC UART controller maintains a two-character length FIFO for receiving data. The data is moved to the buffer after the first character is received into the FIFO and if a receiver FIFO overrun occurs, the channel writes the received character into the internal FIFO. Then the channel writes the received character to the buffer, closes it, sets the OV bit in the buffer descriptor, and generates the RX interrupt if it is enabled. Reception then continues as normal.

Note: The SMC UART controller may occasionally get an overrun error when the line is idle, in which case you should ignore the error.

- Parity Error—When this error occurs, the channel writes the received character to the buffer, closes it, sets the PR bit in the buffer descriptor, and generates the RX interrupt if it is enabled. Reception then continues as normal.
- Idle Sequence Receive Error—An idle is found when one character consisting of all ones is received. Once an idle is received, the channel counts the number of consecutive idle characters. If the count reaches the MAX_IDL value, the buffer is closed, and an RX interrupt is generated. If no receive buffer is open, this event does not generate an interrupt or any status information. The idle counter is reset every time a character is received.
- Framing Error—The SMC UART controller receives this error when it receives a character with no stop bit. When it occurs, the channel writes the received character to the buffer, closes the buffer, sets the FR bit in the buffer descriptor, and generates the RX interrupt if it is enabled. When this error occurs, parity is not checked for the character.
- Break Sequence Error—This error occurs when the SMC UART receiver receives an all-zero character with a framing error. When it occurs, the channel increments the BRKEC entry and generates a maskable BRK interrupt in the SMCE–UART register. The channel also measures the length of the break sequence and stores this value in the BRKLN counter. If the channel was in the middle of buffer processing when the break was received, the buffer is closed with the BR bit in the RX buffer descriptor set and the RX interrupt is generated if it is enabled.

16.11.6.10 SMC UART MODE REGISTER. When the SMC is in UART mode, the 16-bit, memory-mapped, read/write SMC mode register is referred to as the SMC UART mode register (SMCMR–UART). The functionality of bits 8-15 is common to each SMC protocol, but bits 0-7 vary according to the protocol selected by the SM field. This register is cleared by reset.

SMCMR–UART

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RES	CLEN				SL	PEN	PM	RES		SM		DM		TEN	REN
RESET	0	0				0	0	0	0		0		0		0	0
R/W	R/W	R/W				R/W	R/W	R/W	R/W		R/W		R/W		R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA82 (SMC1), 0xA92 (SMC2)															

Bits 0 and 8–9—Reserved

These bits are reserved and should be set to 0.

CLEN—Character Length

This field should be programmed with the total number of bits in the character minus one. The total number of bits in the character is calculated as the sum of 1 (start bit always present) + number of data bits (5 to 14) + number of parity bits (0 or 1) + number of stop bits (1 or 2). For example, for 8 data bits, no parity, and 1 stop bit, the total number of bits in the character is 1 + 8 + 0 + 1 = 10. So, CLEN should be programmed to 9.

The number of data bits in the character ranges from 5 to 14 bits. If the data bit length is less than 8 bits, the most-significant bits of each byte in memory are not used on transmission and are written with zeros on reception. On the other hand, if the data bit length is more than 8 bits, the most-significant bits of each 16-bit word in memory are not used on transmit and are written with zeros on receive.

Note: The total number of bits in the character must never exceed 16. Thus, if you choose a 14-bit data length, you should set SL to one stop bit and not enable parity. If you choose a 13-bit data length and parity is enabled, you should set SL to one stop bit. You should not write the values 0-3 to CLEN or else erratic behavior will occur.

SL—Stop Length

0 = One stop bit.
1 = Two stop bits.

PEN—Parity Enable

0 = No parity.
1 = Parity is enabled for the transmitter and receiver as determined by the PM bit.

PM—Parity Mode

0 = Odd parity.
1 = Even parity.

SM—SMC Mode

00 = GCI or SCIT support.
01 = Reserved.
10 = UART (must be selected for SMC UART operation).
11 = Totally Transparent operation.

DM—Diagnostic Mode

00 = Normal operation.
01 = Local loopback mode.
10 = Echo mode.
11 = Reserved.

TEN—SMC Transmit Enable

0 = SMC transmitter disabled.
1 = SMC transmitter enabled.

REN—SMC Receive Enable

0 = SMC receiver disabled.
1 = SMC receiver enabled.

16.11.6.11 SMC UART RECEIVE BUFFER DESCRIPTOR. Using the buffer descriptors, the communication processor module reports information about the received data on a per-buffer basis. It then closes the current buffer, generates a maskable interrupt, and starts receiving data into the next buffer after one of the following events occurs:

- An error is received while a message is being processed.
- A full receive (RX) buffer is detected.
- A programmable number of consecutive idle characters are received.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	**E**	RES	**W**	**I**	RESERVED		**CM**	ID	RESERVED		BR	FR	PR	RES	OV	RES
OFFSET + 2	DATA LENGTH															
OFFSET + 4	**RX DATA BUFFER POINTER**															
OFFSET + 6																

NOTE: You are only responsible for initializing the items in bold.

E—Empty

0 = The data buffer associated with this RX buffer descriptor is filled with received data or data reception is aborted due to an error condition. The core is free to examine or write to any fields of this RX buffer descriptor. The communication processor module does not use this buffer descriptor as long as the E bit is zero.

1 = The data buffer associated with this buffer descriptor is empty or reception is currently in progress. This RX buffer descriptor and its associated receive buffer are owned by the communication processor module. Once the E bit is set, the core should not write any fields of this RX buffer descriptor.

Bits 1, 4–5, 8–9, 13, and 15—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the RX buffer descriptor table.

1 = This is the last buffer descriptor in the RX buffer descriptor table. After this buffer is used, the communication processor module receives incoming data into the first buffer descriptor that RBASE points to in the table. The number of RX buffer descriptors in this table is programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is filled.

1 = The RX bit in the event register is set when this buffer is completely filled by the communication processor module, indicating the need for the core to process the buffer. The RX bit can cause an interrupt if it is enabled.

CM—Continuous Mode

0 = Normal operation.

1 = The E bit is not cleared by the communication processor module after this buffer descriptor is closed, thus allowing the associated data buffer to be automatically overwritten next time the communication processor module accesses this buffer descriptor. However, the E bit is cleared if an error occurs during reception, regardless of how the CM bit is set.

ID—Buffer Closed on Reception of Idles

This bit indicates that the buffer has closed because a programmable number of consecutive idle sequences have been received. The communication processor module writes this bit after the received data is in the associated data buffer.

BR—Buffer Closed on Reception of Break

This bit indicates that the buffer has closed because a break sequence has been received. The communication processor module writes this bit after the received data is in the associated data buffer.

0 = No break sequence is received.
1 = Buffer closed upon reception of a break sequence.

FR—Framing Error

This bit indicates that a character with a framing error has been received and located in the last byte of this buffer. A framing error is a character without a stop bit. A new receive buffer is used to receive additional data. The communication processor module writes this bit after the received data is in the associated data buffer.

PR—Parity Error

This bit indicates that a character with a parity error has been received and located in the last byte of this buffer. A new receive buffer is used to receive additional data. The communication processor module writes this bit after the received data is in the associated data buffer.

OV—Overrun

This bit indicates that a receiver overrun has occurred during message reception. The communication processor module writes this bit after the received data is in the associated data buffer.

DATA LENGTH

This field represents the number of octets the communication processor module writes into this buffer descriptor data buffer. After the data is received in the associated data buffer, the communication processor module writes this field once the buffer descriptor closes.

Note: The actual amount of memory allocated for this buffer should be greater than or equal to the MRBLR entry.

RX DATA BUFFER POINTER

This field always points to the first location of the associated data buffer and must be even. The buffer can reside in internal or external memory. The communication processor module writes this bit after the received data is in the associated data buffer.

Figure 16-116 illustrates an example of the RX buffer descriptor process. It shows the resulting state of the RX buffer descriptors after they receive 10 characters, an idle period, and five characters (one with a framing error). The example assumes that MRBLR = 8 in the SMC parameter RAM.

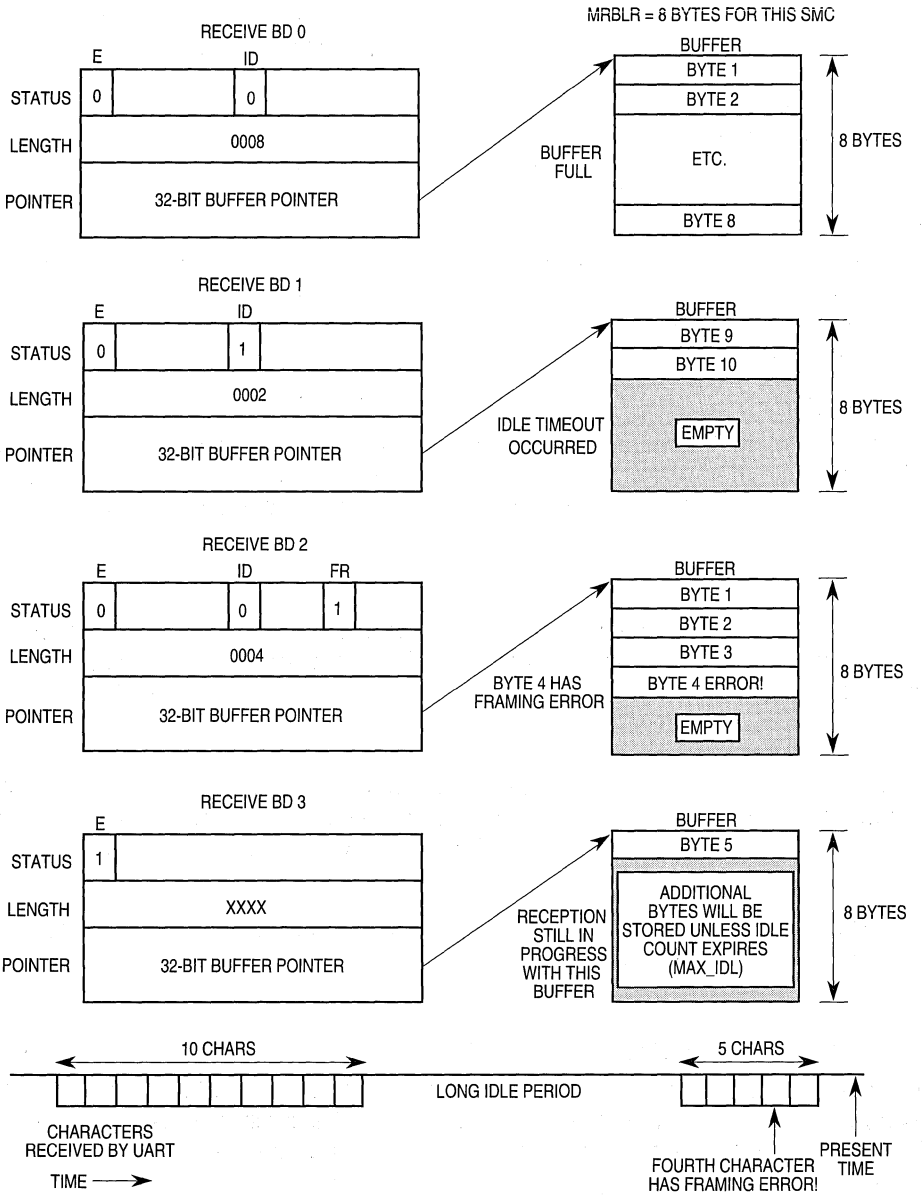


Figure 16-116. SMC UART Receive Buffer Descriptor Example

16.11.6.12 SMC UART TRANSMIT BUFFER DESCRIPTOR. Data is sent to the communication processor module for transmission on an SMC channel by arranging it in buffers referenced by the channel's transmit (TX) buffer descriptor ring. Using the buffer descriptors, the communication processor module confirms transmission or indicates error conditions so that the processor knows the buffers have been serviced.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	R	RES	W	I	RESERVED		CM	P	RESERVED							
OFFSET + 2	**DATA LENGTH**															
OFFSET + 4	**TX DATA BUFFER POINTER**															
OFFSET + 6																

NOTE: You are only responsible for initializing the items in bold.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

R—Ready

0 = The data buffer associated with this buffer descriptor is not ready for transmission, but you are free to manipulate this buffer descriptor or its associated data buffer. The communication processor module clears this bit after the buffer has been transmitted or an error condition is encountered.

1 = The data buffer, which you prepare for transmission, is not transmitted yet or is currently being transmitted. You cannot write any fields of this buffer descriptor once this bit is set.

Bits 1, 4–5, and 8–15—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the TX buffer descriptor table.

1 = This is the last buffer descriptor in the TX buffer descriptor table. After this buffer is used, the communication processor module receives incoming data into the first buffer descriptor that TBASE points to in the table. The number of TX buffer descriptors in this table is programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is serviced.
1 = The TX bit in the SMCE–UART register is set when this buffer is serviced. Transmission can cause an interrupt if it is enabled.

CM—Continuous Mode

0 = Normal operation.
1 = The R bit is not cleared by the communication processor module after this buffer descriptor is closed, thus allowing the associated data buffer to be automatically retransmitted next time the communication processor module accesses this buffer descriptor.

P—Preamble

0 = No preamble sequence is sent.
1 = The SMC UART controller sends one all-ones character before it sends the data so that the other end detects an idle line before the data is received. If this bit is set and the data length of this buffer descriptor is zero, only a preamble is sent.

DATA LENGTH

This field represents the number of octets that the communication processor module should transmit from this buffer descriptor data buffer. However, it is never modified by the communication processor module. Normally, this value should be greater than zero, but it can be equal to zero with the P bit set if only a preamble is sent.

If the number of data bits in the SMC UART character is greater than 8, then the data length should be even. For example, to transmit three UART characters of 8-bit data, 1 start, and 1 stop, the DATA LENGTH field should be initialized to 3. However, to transmit three SMC UART characters of 9-bit data, 1 start, and 1 stop, the DATA LENGTH field should be initialized to 6, since the three 9-bit data fields occupy three half-words in memory (the 9 least-significant bits of each half-word).

TX DATA BUFFER POINTER

This field always points to the first location of the associated data buffer. It can be even or odd, unless the number of actual data bits in the SMC UART character is greater than 8 bits, in which this field is even. For instance, the pointer to 8-bit data, 1 start, and 1 stop characters can be even or odd, but the pointer to 9-bit data, 1 start, and 1 stop characters must be even. The buffer can reside in internal or external memory.

16.11.6.13 SMC UART EVENT REGISTER. When the SMC is in UART mode, the 8-bit memory-mapped SMC event register is referred to as the SMC UART event (SMCE–UART) register. It is used to generate interrupts and report events recognized by the SMC UART channel. When an event is recognized, the SMC UART controller sets the corresponding bit in this register.

A bit is cleared by writing a 1 (writing a zero has no effect) and more than one bit can be cleared at a time. All unmasked bits must be cleared before the communication processor module clears the internal interrupt request. This register is cleared by reset and can be read at any time. An example of the timing of various events in the SMCE–UART register is illustrated in Figure 16-117.

SMCE –UART

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED	BRKE	RESERVED	BRK	RESERVED	BSY	TX	RX
RESET	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA86 (SMC1), 0xA96 (SMC2)							

Bit 0, 2, and 4—Reserved

These bits are reserved and should be set to 0.

BRKE—Break End

This bit indicates that an end of break sequence has been detected. It occurs no sooner than after one idle bit is received after a break sequence.

BRK—Break Character Received

This bit indicates that a break character has been received. If a very long break sequence occurs, this interrupt only occurs once after the first all-zeros character is received.

BSY—Busy Condition

This bit indicates that a character has been received and discarded due to a lack of buffers. It is be set no sooner than the middle of the last stop bit of the first receive character for which there is no available buffer. Reception continues when an empty buffer is provided.

TX—TX Buffer

This bit indicates that a buffer has been transmitted over the SMC UART channel. It is set once the transmit data of the last character in the buffer is written to the transmit FIFO. You must wait two character times to be sure that the data is completely sent over the transmit pin.

RX—Receive Buffer

This bit indicates that a buffer has been received and its associated RX buffer descriptor is now closed. It is set no sooner than the middle of the last stop bit of the last character that is written to the receive buffer.

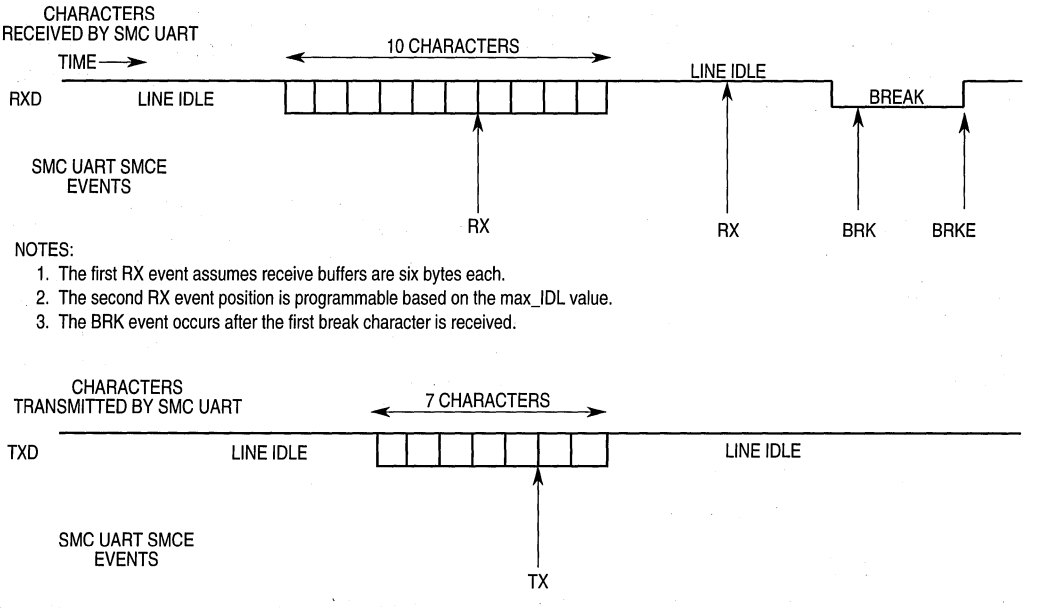


Figure 16-117. SMC UART Interrupt Example

16.11.6.14 SMC UART MASK REGISTER. When the SMC is in UART mode, the 8-bit read/write SMC mask register is referred to as the SMC UART mask (SMCM–UART) register. It has the same bit format as the SMCE–UART register. If a bit in this register is a 1, the corresponding interrupt in the SMCE–UART register is enabled. If the bit is zero, the corresponding interrupt in the SMCE–UART register is masked.

SMCM –UART

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED	BRKE	RESERVED	BRK	RESERVED	BSY	TX	RX
RESET	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA8A (SMC1), 0xA94 (SMC2)							

16.11.6.15 SMC UART CONTROLLER PROGRAMMING EXAMPLE. The following is an initialization sequence for 9,600 baud, 8 data bits, no parity, and 1 stop bit operation of an SMC UART controller assuming a 25MHz system frequency. BRG1 and SMC1 are used.

1. Configure the port B pins to enable SMTXD1 and SMRXD1. Write PBPAR bits 25 and 24 with ones and then PBDIR and PBODR bits 25 and 24 with zeros.
2. Configure the BRG1. Write 0x010144 to BRGC1. The DIV16 bit is not used and the divider is 162 (decimal). The resulting BRG1 clock is 16× the preferred bit rate of the SMC UART controller.
3. Connect the BRG1 clock to SMC1 using the serial interface. Write the SMC1 bit in SIMODE with a 0 and the SMC1CS field in SIMODE register with 0x000.
4. Write RBASE and TBASE in the SMC parameter RAM to point to the RX buffer descriptor and TX buffer descriptor in the dual-port RAM. Assuming one RX buffer descriptor at the beginning of dual-port RAM and one TX buffer descriptor following that RX buffer descriptor, write RBASE with 0x2000 and TBASE with 0x2008.
5. Program the CPCR to execute the **INIT RX AND TX PARAMS** command. Write 0x0091 to the CPCR.
6. Write 0x0001 to the SDCR to initialize the SDMA configuration register.
7. Write 0x18 to the RFCR and TFCR for normal operation.
8. Write MRBLR with the maximum number of bytes per receive buffer. Asume 16 bytes, so MRBLR = 0x0010.
9. Write MAX_IDL with 0x0000 in the SMC UART parameter RAM to disable the MAX_IDL functionality for this example.
10. Clear BRKLN and BRKEC in the SMC UART parameter RAM for the clarity.
11. Set BRKCR to 0x0001, so that if a **STOP TRANSMIT** command is issued, one break character is sent.

12. Initialize the RX buffer descriptor. Assume the RX data buffer is at 0x00001000 in main memory. Write 0xB000 to RX_BD_Status, 0x0000 to RX_BD_Length (not required), and 0x00001000 to RX_BD_Pointer.
13. Initialize the TX buffer descriptor. Assume the TX data buffer is at 0x00002000 in main memory and contains five 8-bit characters. Then write 0xB000 to TX_BD_Status, 0x0005 to TX_BD_Length, and 0x00002000 to TX_BD_Pointer.
14. Write 0xFF to the SMCE–UART register to clear any previous events.
15. Write 0x17 to the SMCM–UART register to enable all possible serial management controller interrupts.
16. Write 0x00000010 to the CIMR so SMC1 can generate a system interrupt. The CICR should also be initialized.
17. Write 0x4820 to SMCMR to configure normal operation (not loopback), 8-bit characters, no parity, 1 stop bit. Notice that the transmitter and receiver are not enabled yet.
18. Write 0x4823 to SMCMR to enable the SMC transmitter and receiver. This additional write ensures that the TEN and REN bits are enabled last.

Note: After 5 bytes are transmitted, the TX buffer descriptor is closed. The receive buffer is closed after 16 bytes are received. Any data received after 16 bytes causes a busy (out-of-buffers) condition since only one RX buffer descriptor is prepared.

16.11.6.16 HANDLING INTERRUPTS IN THE SMC UART CONTROLLER. Follow these steps to handle an interrupt in a serial management controller:

1. Once an interrupt occurs, read the SMCE–UART register to find out what caused it. To clear the SMCE bits, write ones to them.
2. Process the TX buffer descriptor to reuse it if the TX bit is set in the SMCE–UART register. Extract data from the RX buffer descriptor if the RX bit is set in the SMCE–UART. To transmit another buffer, simply set the R bit in the TX buffer descriptor.
3. Clear the SMCx bit in the CISR.
4. Execute the **rfi** instruction.

16.11.7 The SMC in Transparent Mode

The SMC in Transparent mode is also referred to as the SMC Transparent controller. The following is a list of the features that the SMC Transparent controller does not support:

- Independent transmit and receive clocks, unless connected to a TDM channel of the serial interface
- CRC generation and checking
- Full $\overline{\text{RTS}}$, $\overline{\text{CTS}}$, and $\overline{\text{CD}}$ pins. Instead, there is a SMSYN pin for each SMC.
- Ability to transmit data on demand using the TODR
- Receiver/transmitter in transparent mode while executing another protocol
- 4-, 8-, or 16-bit Sync recognition
- Internal DPLL support

The SMC in Transparent mode provides one feature that the SCC2 in Transparent mode did not. The serial management controllers allow a data character length option of 4 to 16 bits, whereas the SCC2 provides 8 or 32 bits, depending on how the RFW bit is set in the GSMR_H, which is described in **Section 16.9.2 The General SCC2 Mode Registers**.

16.11.7.1 FEATURES. The following list summarizes the features of the SMC in Transparent mode:

- Flexible data buffers
- Connects to a TDM bus using the time-slot assigner in the serial interface
- Transmits and receives transparently on its own set of pins using a sync pin to synchronize the beginning of transmission and reception to an external event
- Programmable character length (4-16)
- Reverse data mode
- Continuous transmission and reception modes
- Four commands

16.11.7.2 SMC TRANSPARENT CHANNEL TRANSMISSION PROCESS. The SMC Transparent transmitter is designed to work with almost no intervention from the core. When the core enables the SMC transmitter in transparent mode, it starts transmitting idles. The serial management controllers immediately poll the first buffer descriptor in the transmit channel buffer descriptor ring and once every character time, depending on the character length (every 4 to 16 serial clocks). When there is a message to transmit, a serial management controller fetches the data from memory and starts transmitting the message once synchronization is achieved.

Synchronization can be achieved in two ways. First, when the transmitter is connected to a TDM channel, it can be synchronized to a time-slot. Once the frame sync is received, the transmitter waits for the first bit of its time-slot to occur before it starts transmitting. Data is only transmitted during the time-slots defined by the time-slot assigner. Secondly, when working with its own set of pins, the transmitter starts transmitting when the $\overline{\text{SMSYNx}}$ signal is asserted.

When a buffer descriptor data is completely written to the transmit FIFO, the L bit is checked and if it is set, a serial management controller writes the message status bits into the buffer descriptor and clears the R bit. It then starts transmitting idles. When the end of the current buffer descriptor is reached and the L bit is not set, only the R bit is cleared. In both cases, an interrupt is issued according to the I bit in the buffer descriptor. By appropriately setting the I bit in each buffer descriptor, interrupts can be generated after each buffer, a specific buffer, or each block is transmitted. The serial management controller then proceeds to the next buffer descriptor in the table. If no additional buffers have been presented to the serial management controller for transmission and the L bit was cleared, an underrun is detected and the serial management controller begins transmitting idles.

If the CM bit is set in the TX buffer descriptor, the R bit is not cleared, thus allowing the associated data buffer to be automatically retransmitted next time the communication processor module accesses this data buffer. For instance, if a single TX buffer descriptor is initialized with the CM bit and the W bit set, the data buffer is continuously transmitted until you clear the R bit of the buffer descriptor.

16.11.7.3 SMC TRANSPARENT CHANNEL RECEPTION PROCESS. When the core enables the SMC receiver in transparent mode, it waits for synchronization before receiving data. Once synchronization is achieved, the receiver transfers the incoming data into memory according to the first RX buffer descriptor in the ring. Synchronization can be achieved in two ways. First, when the receiver is connected to a TDM channel, it can be synchronized to a time-slot. Once the frame sync is received, the receiver waits for the first bit of its time-slot to occur before reception begins. Data is only received during the time-slots defined by the time-slot assigner. Secondly, when working with its own set of pins, the receiver starts reception when the $\overline{\text{SMSYNx}}$ signal is asserted.

When the data buffer is filled, a SMC Transparent controller clears the E bit in the buffer descriptor and generates an interrupt if the I bit in the buffer descriptor is set. If the incoming data exceeds the length of the data buffer, a serial management controller fetches the next buffer descriptor in the table and, if it is empty, continues transferring data to this buffer descriptor associated data buffer. If the CM bit is set in the RX buffer descriptor, the E bit is not cleared, thus allowing the associated data buffer to be automatically overwritten next time the communication processor module accesses this data buffer.

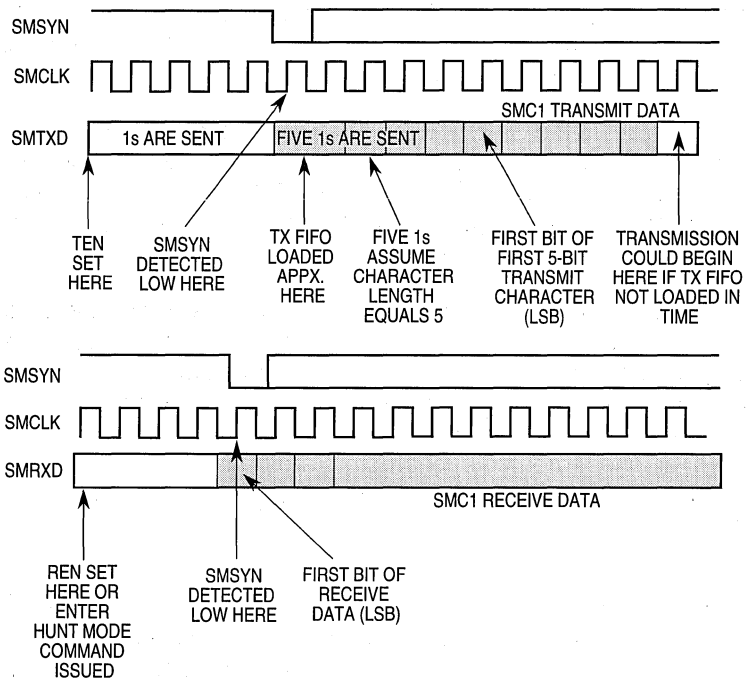
16.11.7.4 USING THE $\overline{\text{SMSYNx}}$ PIN FOR SYNCHRONIZATION. The $\overline{\text{SMSYNx}}$ pin offers a method to externally synchronize a SMC Transparent channel. This method differs somewhat from the synchronization options available in the SCC2 and should be studied carefully. See Figure 16-118 for an example.

Once the REN bit is set in the SMCMR, the first rising edge of the SMCLK signal that finds the $\overline{\text{SMSYNx}}$ pin low causes the SMC receiver to achieve synchronization. Data starts being received or latched on the same rising edge of SMCLK that latched $\overline{\text{SMSYNx}}$. This is the first bit of data received. The receiver never loses synchronization again, regardless of the state of $\overline{\text{SMSYNx}}$, until you clear the REN bit.

Once the TEN bit is set in the SMCMR, the first rising edge of the SMCLK signal that finds the $\overline{\text{SMSYNx}}$ pin low causes the SMC transmitter to achieve synchronization. The SMC transmitter begins transmitting ones asynchronously from the falling edge of $\overline{\text{SMSYNx}}$. After one character of ones is transmitted, if the transmit FIFO is loaded (the TX buffer descriptor is ready with data), data starts being transmitted on the next falling edge of SMCLK after some multiple of all-ones (preamble) characters are transmitted. If the transmit FIFO is loaded at some later time, the data starts transmitting after some multiple number of all-ones characters is transmitted.

Note: Regardless of whether the transmitter or receiver uses the $\overline{\text{SMSYNx}}$ signal, it must make glitch-free transitions from high to low or low to high. Glitches on $\overline{\text{SMSYNx}}$ can cause a serial management controller to behave erratically.

The transmitter never loses synchronization again, regardless of the state of $\overline{\text{SMSYNx}}$, until you clear the TEN bit or issue the **ENTER HUNT MODE** command.



NOTES:
1. SMCLK is an internal clock derived from an external CLKPIN or a baud rate generator.
2. This example shows the SMC receiver and transmitter enabled separately. If the REN and TEN bits were set at the same time, a single falling edge of SMSYN would synchronize both.

Figure 16-118. $\overline{\text{SMSYNx}}$ Pin Synchronization

If both the REN and TEN bits are set in the SMCMR, the first falling edge of the $\overline{\text{SMSYNx}}$ pin causes both the transmitter and receiver to achieve synchronization. To resynchronize the transmitter, the SMC transmitter can be disabled and reenabled and the $\overline{\text{SMSYNx}}$ pin can be used again to resynchronize the transmitter itself. Refer to **Section 16.11.5 Disabling the SMCs On-the-Fly** for a description of how to safely disable and reenable a serial management controller. Simply clearing and setting the TEN bit may not be sufficient. The receiver can also be resynchronized this way.

16.11.7.5 USING THE TIME-SLOT ASSIGNER FOR SYNCHRONIZATION. The time-slot assigner offers a method to internally synchronize a SMC Transparent channel without using the $\overline{\text{SMSYNx}}$ pin. This method is similar to that of the $\overline{\text{SMSYNx}}$ pin, except that the synchronization event is not the falling edge of the $\overline{\text{SMSYNx}}$ signal, but the first time-slot for this SMC receiver/transmitter after the frame sync indication. Refer to **Section 16.7 The Serial Interface with Time-Slot Assigner** for further information about configuring time-slots for the SMCs and SCC2.

The time-slot assigner allows the SMC receiver and transmitter to be enabled simultaneously and synchronized separately, a capability that the $\overline{\text{SMSYNx}}$ pin does not provide. Refer to Figure 16-119 for an example of synchronization using the time-slot assigner.

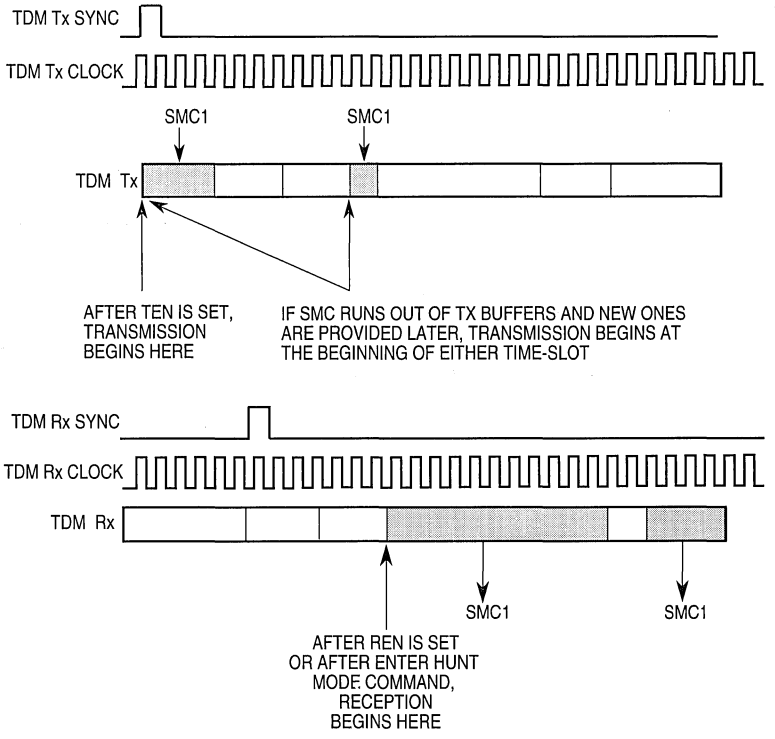


Figure 16-119. Time-Slot Assigner Synchronization

Once the REN bit is set in the SMCMR, the first time-slot after frame sync causes the SMC receiver to achieve synchronization. Data is received immediately, but only during the defined receive time-slots. The receiver continues receiving data during its defined time-slots until you clear the REN bit. If the **ENTER HUNT MODE** command is issued, the receiver loses synchronization, closes the current buffer, and resynchronizes to the first time-slot after the frame sync.

Once the TEN bit is set in SMCMR, an SMC Transparent controller waits for the transmit FIFO to be loaded before trying to achieve synchronization. Once the transmit FIFO is loaded, synchronization and transmission begins depending on the following situations:

- If a buffer is made ready when an SMC Transparent controller is enabled, then the first byte will be placed in time-slot 1 if the CLEN field in SMCMR is set to 8 and slot 2 if CLEN is set to 16.
- If a buffer has an SMC Transparent controller enabled, then the first byte in the next buffer can appear in any time-slot associated with this channel.
- If a buffer is ended with the L bit set, then the next buffer can appear in any time-slot associated with this channel.

If an SMC Transparent controller runs out of transmit buffers and a new transmit buffer is provided later, idles are transmitted during the gap between data buffers. Data transmission from the later data buffer begins at the beginning of an SMC Transparent controller time-slot, but not necessarily the first time-slot after the frame sync. So if you want to maintain a certain bit alignment beginning with the first time-slot, make sure that at least one TX buffer descriptor is always ready and that no underrun occurs. Otherwise, the SMC Transparent transmitter should be disabled and reenabled. Refer to **Section 16.11.5 Disabling the SMCs On-the-Fly** for a description of how to safely disable and reenable the SMC Transparent controller. Simply clearing TEN and setting TEN may not be sufficient.

16.11.7.6 SMC TRANSPARENT CONTROLLER PARAMETER RAM MEMORY MAP. There is no protocol-specific parameter RAM for the SMC Transparent controller. Only the general SMC parameter RAM is used, which is discussed in more detail in **Section 16.11.4 SMC General Parameter RAM Memory Map**.

16.11.7.7 SMC TRANSPARENT COMMANDS. You can program the CPM command register (CPCR) with the following commands to transmit data.

- **STOP TRANSMIT**—After the hardware or software is reset and the channel is enabled in the SMCM–Transparent register, the channel is in the transmit enable mode and starts polling the first buffer descriptor in the table. This command disables the transmission of frames on the transmit channel. If the transparent controller receives this command while transmitting a frame, it stops after the contents of the FIFO are transmitted (up to 2 characters). The TBPTR is not advanced to the next buffer descriptor, no new buffer descriptor is accessed, and no new buffers are transmitted for this channel. The transmitter sends idles until the **RESTART TRANSMIT** command is issued.
- **RESTART TRANSMIT**—This command is used to begin or continue transmission from the current TBPTR in the channel's TX buffer descriptor table. When the channel receives this command, it starts polling the R bit in the TX buffer descriptor. A serial management controller expects this command after a **STOP TRANSMIT** command is issued and the channel in the SMCMR is disabled or after a transmitter error occurs.
- **INIT TX PARAMETERS**—This command initializes all the transmit parameters in this serial channel parameter RAM to their reset state and should only be issued when the transmitter is disabled. The **INIT TX AND RX PARAMS** command can also be used to reset the transmit and receive parameters.

You can program the CPCR with the following commands to receive data.

- **ENTER HUNT MODE**—This command forces a serial management controller to close the current receive buffer descriptor if it is currently being used and to use the next buffer descriptor in the list for any subsequently received data. If a serial management controller is not in the process of receiving data, the buffer is not closed. Additionally, this command causes the receiver to wait for a resynchronization before further reception continues.
- **CLOSE RX BD**—This command is used to force a serial management controller to close the current receive buffer descriptor if it is being used and to use the next buffer descriptor in the list for any subsequently received data. If a serial management controller is not in the process of receiving data, no action is taken by this command.
- **INIT RX PARAMETERS**—This command initializes all the receive parameters in this serial channel's parameter RAM to their reset state. It should only be issued when the receiver is disabled. The **INIT TX AND RX PARAMS** command can also be used to reset the receive and transmit parameters.

16.11.7.8 SMC TRANSPARENT CONTROLLER ERRORS. The serial management controllers report message reception and transmission error conditions using the channel buffer descriptors and the SMCE–Transparent register. The following transmission errors can be detected by the SMC Transparent controller.

- Underrun Error—When this error occurs, the channel stops transmitting the buffer, closes it, sets the UN bit in the buffer descriptor, and generates the TXE interrupt if it is enabled. The channel resumes transmission after it receives the **RESTART TRANSMIT** command. Underrun cannot occur between frames.
- Overrun Error—A serial management controller maintains an internal FIFO for receiving data. The communication processor module begins programming the SDMA channel if the data buffer is in external memory when the first character is received into the FIFO. If a FIFO overrun occurs, a serial management controller writes the received data character to the internal FIFO over the previously received character. The previous character and its status bits are lost. Then the channel closes the buffer, sets the OV bit in the buffer descriptor, and generates the RX interrupt if it is enabled. Reception then continues as normal.

16.11.7.9 SMC TRANSPARENT MODE REGISTER. When the SMC is in transparent mode, the 16-bit, memory-mapped, read/write SMC mode register is referred to as the SMC transparent mode register (SMCMR–Transparent). The function of bits 8-15 is common to each SMC protocol, but bits 0-7 vary according to the protocol selected by the SM bits of this register.

SMCMR–TRANSPARENT

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RES	CLEN				RES	BS	REVD	RESERVED		SM		DM		TEN	REN
RESET	0	0				0	0	0	0		0		0		0	0
R/W	R/W	R/W				R/W	R/W	R/W	R/W		R/W		R/W		R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA82 (SMC1), 0xA92 (SMC2)															

Bits 0, 5, and 8–9—Reserved

These bits are reserved and should be set to 0.

CLEN—Character Length

This field is programmed with a value between 3 and 15 to obtain 4 to 16 bits per character. If the character length is less than 8 bits, the MSBs of the byte in buffer memory are not used on transmit and are written with zeros on receive. On the other hand, if the character length is more than 8 bits but less than 16 bits, the MSBs of the half-word in buffer memory are not used on transmit and are written with zeros on receive.

Note: You should not write the values 0 to 2 to CLEN or else erratic behavior will occur. Larger character lengths increase the potential performance of the SMC channel and lower the performance impact of other channels. For instance, using 16-bit characters, rather than 8-bit characters is encouraged if 16-bit characters are acceptable in the end application.

BS—Byte Sequence

This bit controls the sequence of byte transmission if the REVD bit is set for a character length greater than 8 bits. It should be set to zero to maintain behavior compatibility with the MC68360 QUICC microprocessor.

0 = Normal mode. This should be selected if the character length is less than or equal to 8 bits.
1 = Transmit lower address byte first.

REVD—Reverse Data

0 = Normal mode.
1 = Reverse the character bit order. The MSB is transmitted first.

SM—SMC Mode

00 = GCI or SCIT support.
01 = Reserved.
10 = UART.
11 = Totally transparent operation. This must be selected for SMC transparent operation.

DM—Diagnostic Mode

00 = Normal operation.
01 = Local loopback mode.
10 = Echo mode.
11 = Reserved.

TEN—SMC Transmit Enable

0 = SMC transmitter disabled.
1 = SMC transmitter enabled.

Note: Once the TEN bit is cleared, it must not be reenabled for at least three serial clocks.

REN—SMC Receive Enable

0 = SMC receiver disabled.
1 = SMC receiver enabled.

16.11.7.10 SMC TRANSPARENT RECEIVE BUFFER DESCRIPTOR. Using receive (RX) buffer descriptors, the communication processor module reports information about the received data for each buffer and closes the current buffer, generates a maskable interrupt, and starts to receive data into the next buffer after one of the following events occurs:

- An overrun error occurs.
- A full receive buffer is detected.
- The **ENTER HUNT MODE** command is issued.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	**E**	RES	**W**	**I**	RESERVED		**CM**	RESERVED							OV	RES
OFFSET + 2	DATA LENGTH															
OFFSET + 4	**RX DATA BUFFER POINTER**															
OFFSET + 6																

NOTE: You are only responsible for initializing the items in bold.

E—Empty

0 = The data buffer associated with this RX buffer descriptor is filled with received data or data reception has been aborted due to an error condition. The core is free to examine or write to any fields of this RX buffer descriptor. The communication processor module does not use this buffer descriptor as long as the E bit is zero.

1 = The data buffer associated with this buffer descriptor is empty or is currently receiving data. This RX buffer descriptor and its associated receive buffer are owned by the communication processor module. Once the E bit is set, the core should not write any fields of this RX buffer descriptor.

Bits 1, 4–5, 7–13, and 15—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the RX buffer descriptor table.
1 = This is the last buffer descriptor in the RX buffer descriptor table. After this buffer is used, the communication processor module receives incoming data into the first buffer descriptor that RBASE points to in the table. The number of RX buffer descriptors in this table is programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is filled.
1 = The RX bit in the SMCE–Transparent register is set when this buffer is completely filled by the communication processor module, thus indicating that the core needs to process the buffer. The RX bit can cause an interrupt if it is enabled.

CM—Continuous Mode

0 = Normal operation.
1 = The E bit is not cleared by the communication processor module after this buffer descriptor is closed, thus allowing the associated data buffer to be automatically overwritten next time the communication processor module accesses this buffer descriptor. However, the E bit is cleared if an error occurs during reception, regardless of how the CM bit is set.

OV—Overrun

This bit indicates that a receiver overrun has occurred during message reception. The communication processor module writes this bit after the received data is placed into the associated data buffer.

DATA LENGTH

This field represents the number of octets that the communication processor module writes into this buffer descriptor data buffer. It is written only once by the communication processor module as the buffer is closed. The communication processor module writes these bits after the received data is placed into the associated data buffer.

Note: The actual amount of memory allocated for this buffer should be greater than or equal to the MRBLR entry.

RX DATA BUFFER POINTER

This field always points to the first location of the associated data buffer, must be even, and can reside in internal or external memory. The communication processor module writes these bits after the received data is placed into the associated data buffer.

16.11.7.11 SMC TRANSPARENT TRANSMIT BUFFER DESCRIPTOR. Data is sent to the communication processor module for transmission on an SMC channel by arranging it in buffers referenced by the channel's transmit (TX) buffer descriptor table. Using the buffer descriptors, the communication processor module confirms transmission or indicates error conditions so that the processor knows the buffers have been serviced.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	**R**	RES	**W**	**I**	**L**	RES	**CM**	RESERVED							UN	RES
OFFSET + 2	**DATA LENGTH**															
OFFSET + 4	**TX DATA BUFFER POINTER**															
OFFSET + 6																

NOTE: You are only responsible for initializing the items in bold.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

R—Ready

0 = The data buffer associated with this buffer descriptor is not ready for transmission and you are free to manipulate it or its associated data buffer. The communication processor module clears this bit after the buffer is transmitted or after an error condition is encountered.

1 = The data buffer, which you prepare for transmission, is not transmitted yet or is currently being transmitted. You cannot write any fields of this buffer descriptor once this bit is set.

Bits 1, 5, 7–13, and 15—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the TX buffer descriptor table.

1 = This is the last buffer descriptor in the TX buffer descriptor table. After this buffer is used, the communication processor module receives incoming data into the first buffer descriptor that TBASE points to in the table. The number of TX buffer descriptors in this table is programmable and determined by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is serviced.
1 = The TX and TXE bits in the SMCE–Transparent register are set when this buffer is serviced. TX and TXE can cause interrupts if they are enabled.

L— Last in Message

0 = The last byte in the buffer is not the last byte in the transmitted transparent frame. Data from the next transmit buffer (if ready) is transmitted immediately following the last byte of this buffer.
1 = The last byte in this buffer is the last byte in the transmitted transparent frame. After this buffer is transmitted, the transmitter requires synchronization before the next buffer is transmitted.

CM—Continuous Mode

0 = Normal operation.
1 = The communication processor module does not clear the R bit after this buffer descriptor is closed, thus allowing the associated data buffer to be automatically retransmitted next time the communication processor module accesses this buffer descriptor. However, the R bit will be cleared if an error occurs during transmission, regardless of how the CM bit is set.

UN—Underrun

This bit indicates that a serial management controller has encountered a transmitter underrun condition while transmitting the associated data buffer.

DATA LENGTH

This field represents the number of octets that the communication processor module should transmit from this buffer descriptor data buffer and it is never modified by the communication processor module. This field can be even or odd, but if the number of bits in the transparent character is greater than 8, this field should be even. For example, to transmit three transparent 8-bit characters, this field should be initialized to 3. However, to transmit three transparent 9-bit characters, this field should be initialized to 6 since the three 9-bit characters occupy three half-words in memory.

TX DATA BUFFER POINTER

This field always points to the first byte of the associated data buffer. They can be even or odd, unless the character length is greater than 8 bits, in which case this field must be even. For instance, the pointer to 8-bit transparent characters can be even or odd, but the pointer to 9-bit transparent characters must be even. The buffer can reside in internal or external memory.

16.11.7.12 SMC TRANSPARENT EVENT REGISTER. When the SMC is in transparent mode, the 8-bit memory-mapped SMC event register is referred to as the SMC transparent event (SMCE–Transparent) register. It is used to generate interrupts and report events recognized by the SMC channel. When an event is recognized, the serial managment controller sets the corresponding bit in this register. Interrupts generated by this register can be masked in the SMCM–Transparent register.

A bit is cleared by writing a 1 (writing a zero has no effect) and more than one bit can be cleared at a time. All unmasked bits must be cleared before the communication processor module clears the internal interrupt request. This register is cleared at reset and can be read at any time.

SMCE–TRANSPARENT

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED			TXE	RES	BSY	TX	RX
RESET	0			0	0	0	0	0
R/W	R/W			R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA86 (SMC1), 0xA96 (SMC2)							

Bits 0–2 and 4—Reserved

These bits are reserved and should be set to 0.

TXE—TX Error

This bit indicates that an underrun error has occurred on the transmitter channel.

BSY—Busy Condition

This bit indicates that a character has been received and discarded due to a lack of buffers. Reception begins after a new buffer is provided. You can execute an **ENTER HUNT MODE** command to make the receiver wait for resynchronization.

TX—TX Buffer

This bit indicates that a buffer has been transmitted. If the L bit of the TX buffer descriptor is set, this bit is set when the last data character starts being transmitted and you must wait one character time to be sure that the data is completely sent over the transmit pin. If the L bit of the TX buffer descriptor is cleared, this bit is set when the last data character is written to the transmit FIFO and you must wait two character times to be sure that the data is completely sent over the transmit pin.

RX—RX Buffer

This bit indicates that a abuffer has been received on the SMC channel and its associated RX buffer descriptor is now closed. This bit is set after the last character is written to the buffer.

16.11.7.13 SMC TRANSPARENT MASK REGISTER. When the SMC is in transparent mode, the 8-bit read/write SMC mask register is referred to as the SMC transparent mask (SMCM–Transparent) register. It has the same bit format as the SMCE–Transparent register. If a bit in this register is a 1, the corresponding interrupt in theSMCE–Transparent register is enabled. If the bit is zero, the corresponding interrupt is masked.

SMCM–TRANSPARENT

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED			TXE	RES	BSY	TX	RX
RESET	0			0	0	0	0	0
R/W	R/W			R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA8A (SMC1), 0xA9A (SMC2)							

16.11.7.14 SMC TRANSPARENT NMSI PROGRAMMING EXAMPLE. The following is an example initialization sequence for SMC1 transparent channel over its own set of pins. The transmit and receive clocks are provided from the CLK3 pin and the SMSYN1 pin is used to obtain synchronization.

1. Configure the port B pins to enable the SMTXD1, SMRXD1, and SMSYN1. Write PBPAR bits 25, 24, and 23 with ones and then PBDIR and PBODR bits 25, 24, and 23 with zeros.
2. Configure the port A pins to enable CLK3. Write PAPAR bit 5 with a one and PADIR bit 5 with a zero. The other functions of this pin are the timers or the time-slot assigner. These alternate functions cannot be used on this pin.
3. Connect the CLK3 clock to SMC1 using the serial interface. Write the SMC1 bit in the SIMODE register with a 0 and the SMC1CS field in the SIMODE register with 110.
4. Write RBASE and TBASE in the SMC parameter RAM to point to the RX buffer descriptor and TX buffer descriptor in the dual-port RAM. Assuming one RX buffer descriptor at the beginning of the dual-port RAM and one TX buffer descriptor following that RX buffer descriptor, write RBASE with 0x2000 and TBASE with 0x2008.
5. Program the CPCR to execute the **INIT RX AND TX PARAMS** command. Write 0x0091 to the CPCR.
6. Write 0x0001 to the SDCR to initialize the SDMA configuration register.
7. Write 0x18 to RFCR and TFCR for normal operation.
8. Write MRBLR with the maximum number of bytes per receive buffer. Assume 16 bytes, so MRBLR = 0x0010.
9. Initialize the RX buffer descriptor and assume the RX data buffer is at 0x00001000 in main memory. Write 0xB000 to RX_BD_Status, 0x0000 to RX_BD_Length (optional), and 0x00001000 to RX_BD_Pointer.

10. Initialize the TX buffer descriptor and assume the TX data buffer is at 0x00002000 in main memory and contains five 8-bit characters. Write 0xB000 to TX_BD_Status, 0x0005 to TX_BD_Length, and 0x00002000 to TX_BD_Pointer.
11. Write 0xFF to the SMCE–Transparent register to clear any previous events.
12. Write 0x13 to the SMCM–Transparent register to enable all possible serial management controller interrupts.
13. Write 0x00000010 to the CIMR to allow SMC1 to generate a system interrupt. The CICR should also be initialized.
14. Write 0x3830 to the SMCMR to configure 8-bit characters, unreversed data, and normal operation (not loopback). Notice that the transmitter and receiver have not been enabled yet.
15. Write 0x3833 to the SMCMR to enable the SMC transmitter and receiver. This additional write ensures that the TEN and REN bits are enabled last.

Note: After 5 bytes are transmitted, the TX buffer descriptor is closed and after 16 bytes are received the receive butter is closed too. Any data received after 16 bytes causes a busy (out-of-buffers) condition since only one RX buffer descriptor is prepared.

16.11.7.15 SMC TRANSPARENT TSA PROGRAMMING EXAMPLE. The following is an example initialization sequence for the SMC1 transparent channel over the time-slot assigner. It is assumed that the time-slot assigner and the TDM pins have already been set up to route time-slot data to the SMC transmitter and receiver. Refer to **Section 16.7 The Serial Interface with Time-Slot Assigner** for examples of how to configure the time-slot assigner. The transmit and receive clocks and synchronization signals are provided internally from the time-slot assigner.

1. Write RBASE and TBASE in the SMC parameter RAM to point to the RX buffer descriptor and TX buffer descriptor in the dual-port RAM. Assuming one RX buffer descriptor at the beginning of the dual-port RAM and one TX buffer descriptor following that RX buffer descriptor, write RBASE with 0x2000 and TBASE with 0x2008.
2. Program the CPCR to execute the **INIT TX AND RX PARAMS** command. Write 0x0091 to the CPCR.
3. Write 0x0001 to the SDCR to initialize the SDMA configuration register.
4. Write 0x18 to RFCR and TFCR for normal operation.
5. Write MRBLR with the maximum number of bytes per receive buffer. Assume 16 bytes, so MRBLR = 0x0010.
6. Initialize the RX buffer descriptor and assume the RX data buffer is at 0x00001000 in main memory. Write 0xB000 to RX_BD_Status, 0x0000 to RX_BD_Length (optional), and 0x00001000 to RX_BD_Pointer.

7. Initialize the TX buffer descriptor and assume the TX data buffer is at 0x00002000 in main memory and contains five 8-bit characters. Write 0xB000 to TX_BD_Status, 0x0005 to TX_BD_Length, and 0x00002000 to TX_BD_Pointer.
8. Write 0xFF to the SMCE–Transparent register to clear any previous events.
9. Write 0x13 to the SMCM–Transparent register to enable all possible serial management controller interrupts.
10. Write 0x00000010 to the CIMR so that SMC1 can generate a system interrupt. The CICR should also be initialized.
11. Write 0x3830 to the SMCMR–Transparent to configure 8-bit characters, unreversed data, and normal operation (not loopback). Notice that the transmitter and receiver are not enabled yet.
12. Write 0x3833 to the SMCMR–Transparent to enable the SMC transmitter and receiver. This additional write ensures that the TEN and REN bits are enabled last.

16.11.7.16 HANDLING INTERRUPTS IN THE SMC. Follow these steps to handle an interrupt in the serial management controller:

1. Once an interrupt occurs, read the SMCE register to find out what caused the interrupts. The SMCE bits are usually cleared at this time.
2. Process the TX buffer descriptor to reuse it if the TX bit is set in the SMCE–Transparent register. Extract data from the RX buffer descriptor if the RX bit is set in the SMCE–Transparent. To transmit another buffer, simply set the R bit in the RX buffer descriptor.
3. Clear the SMC1 bit in the CISR.
4. Execute the **rfi** instruction.

16.11.8 The SMC in GCI Mode

The serial management controller can be used to control the circuit interface and monitor channels of the general circuit interface (GCI) frame. When using the SCIT configuration of a general circuit interface, one serial management controller can handle SCIT channel 0, and the other serial management controller can handle SCIT channel 1. The main features of the SMC in GCI mode are as follows:

- Each SMC channel supports the circuit interface and monitor channels of the GCI (IOM-2) in ISDN applications
- Two serial management controllers support the two sets of circuit interface and monitor channels in SCIT channels 0 and 1
- Full-duplex operation
- Local loopback and echo capability for testing

To use the SMC GCI channels properly, the time-slot assigner in the serial interface must be configured to route the monitor and circuit interface channels to the serial management controller you prefer. Refer to **Section 16.7 The Serial Interface with Time-Slot Assigner** for more details on how to program this configuration. The SMC in GCI mode is also referred to as the SMC GCI controller.

16.11.8.0.1 SMC GCI Monitor Channel Transmission Process. The monitor channel 0 is used to exchange data with a layer 1 device (reading and writing internal registers and transferring of the S and Q bits). Monitor channel 1 is used for programming and controlling voice/data modules, such as CODECs. The core writes the data byte into the transmit (TX) buffer descriptor. The serial management controller transmits the data on the monitor channel and handles the A and E control bits according to the GCI monitor channel protocol. You can issue the **TIMEOUT** command to solve deadlocks when errors in the A and E bit occur on the data line.

16.11.8.0.2 SMC GCI Monitor Channel Reception Process. The serial management controller receives data and handles the A and E control bits according to the GCI monitor channel protocol. When the communication processor module stores a received data byte in the SMC receive (RX) buffer descriptor, a maskable interrupt is generated. You can issue the **TRANSMIT ABORT REQUEST** command and the MPC823 transmits an abort request on the E bit.

16.11.8.1 HANDLING THE SMC CIRCUIT INTERFACE CHANNEL. The circuit interface channel is used to control the layer 1 device. The layer 2 device in the TE sends commands and receives indication to or from the upstream layer 1 device via circuit interface channel 0. In the SCIT configuration, circuit interface channel 1 is used to convey real-time status information between the layer 2 device and nonlayer 1 peripheral devices (CODECs).

16.11.8.1.1 SMC GCI Circuit Interface Channel Transmission Process. The core writes the data byte into the circuit interface TX buffer descriptor and the serial management controller transmits the data continuously on the circuit interface channel to the physical layer device.

16.11.8.1.2 SMC GCI Circuit Interface Channel Reception Process. The SMC receiver continuously monitors the circuit interface channel and when it recognizes a change in the data and this value is received in two successive frames, it is interpreted as valid data. This is referred to as the double last-look method. The received data byte is stored by the communication processor module in the circuit interface RX buffer descriptor and a maskable interrupt is generated. If the serial management controller is configured to support SCIT channel 1, the double last-look method is not used.

16.11.8.2 SMC GCI PARAMETER RAM MEMORY MAP. The SMC GCI parameter RAM area begins at the same offset from each SMC base area. The SMC in GCI mode has a very different parameter RAM memory map than the SMC in UART or transparent mode. In GCI mode, the general-purpose parameter RAM contains the buffer descriptors, instead of pointers, to the buffer descriptors. Compare Table 16-38 with Table 16-36 to see the differences. The SMC in GCI mode contains no protocol-specific parameter RAM.

Table 16-38. SMC GCI Parameter RAM Memory Map

ADDRESS	NAME	WIDTH	DESCRIPTION
SMC Base + 00	**M_RXBD**	Half-word	Monitor Channel RX BD
SMC Base + 02	**M_TXBD**	Half-word	Monitor Channel TX BD
SMC Base + 04	**CI_RXBD**	Half-word	Circuit Interface Channel RX BD
SMC Base + 06	**CI_TXBD**	Half-word	Circuit Interface Channel TX BD
SMC Base + 08	Temp1	Half-word	
SMCBase + 0A	Temp2	Half-word	
SMC Base + 0C	Temp3	Half-word	
SMC Base + 0E	Temp4	Half-word	

NOTE: You are only responsible for initializing the items in bold.
SMC Base = (IMMR & 0xFFFF0000) + 0x3E80 (SMC1) and 0x3F80 (SMC2).

- M_RXBD—The SMC monitor channel receive buffer descriptor is used by the communication processor module to report information about the monitor channel receive byte.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	**E**	L	**RE**	**MS**	RESERVED				DATA							

NOTE: You are only responsible for initializing the items in bold.

E—Empty

When a serial management controller uses the monitor channel protocol, it waits until the core sets this bit before acknowledging the monitor channel data.

0 = The communication processor module clears this bit to indicate that the data byte associated with this buffer descriptor is now available to the core.
1 = The core sets this bit to indicate that the data byte associated with this buffer descriptor has been read.

L—Last (EOM)

This bit is only valid when a serial management controller implements the monitor channel protocol and is set when the EOM indication is received on the E bit. When this bit is set, the data byte is invalid.

ER—Error Condition

This bit is only valid when a serial management controller implements the monitor channel protocol and is set when an error condition occurs on the monitor channel protocol. A new byte is transmitted before a serial management controller acknowledges the previous byte.

MS—Data Mismatch

This bit is only valid when a serial management controller implements the monitor channel protocol. It is set when two different consecutive bytes are received and it is cleared when the last two consecutive bytes match. A serial management controller waits for the reception of two identical consecutive bytes before writing new data to the RX buffer descriptor.

Bits 4–7—Reserved

These bits are reserved and should be set to 0.

DATA—Data

This field contains the monitor channel data byte that a serial management controller received.

- M_TXBD—The SMC monitor channel transmit buffer descriptor is used by the communication processor module to report information about the monitor channel transmit byte.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	R	L	AR	RESERVED					DATA							

NOTE: You are only responsible for initializing the items in bold.

R—Ready

0 = This bit is cleared by the communication processor module after transmission. The TX buffer descriptor is now available to the core.

1 = The core sets this bit to indicate that the data byte associated with this buffer descriptor is ready for transmission.

L—Last (EOM)

This bit is only valid when a serial management controller implements the monitor channel protocol. When it is set, a serial management controller first transmits the buffer data and then transmits the EOM indication on the E bit.

AR—Abort Request

This bit is only valid when a serial management controller uses the monitor channel protocol and it is set by a serial management controller when an abort request is received on the A bit. The SMC transmitter transmits the EOM on the E bit after an abort request is received.

Bits 3–7—Reserved

These bits are reserved and should be set to 0.

DATA—Data Field

This field contains the data to be transmitted by a serial management controller on the monitor channel.

- C/I_RXBD—The SMC circuit interface channel receive (RX) buffer descriptor is used by the communication processor module to report information about the circuit interface channel receive byte.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	E	RES							C/I DATA						RESERVED	

E—Empty

0 = The communication processor module clears this bit to indicate that the data byte associated with this buffer descriptor is now available to the core.

1 = The core set this bit to indicate that the data byte associated with this buffer descriptor has been read.

Note: Additional data received is discarded until the E bit is set.

Bits 1–7 and 14–15—Reserved

These bits are reserved and should be set to 0.

C/I DATA—Command/Indication Data Bits

This field represents a 4-bit data field for circuit interface channel 0 and a 6-bit data field for circuit interface channel 1. It contains the data received from the circuit interface channel. For circuit interface channel 0, bits 10-13 contain the 4-bit data field and bits 8 and 9 are always written with zeros. For circuit interface channel 1, bits 8-13 contain the 6-bit data field.

- C/I_TXBD—The SMC circuit interface channel transmit (TX) buffer descriptor is used by the communication processor module to report information about the circuit interface channel transmit byte.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	R	RESERVED							C/I DATA						RESERVED	

R—Ready

0 = The communication processor module clears this bit after transmission to indicate that the buffer descriptor is now available to the core.

1 = The core sets this bit to indicate that the data associated with this buffer descriptor is ready for transmission.

Bits 1–7 and 14–15—Reserved

These bits are reserved and should be set to 0.

C/I DATA—Command/Indication Data Bits

This field represents a 4-bit data field for circuit interface channel 0 and a 6-bit data field for circuit interface channel 1. It contains the data to be transmitted onto the circuit interface channel. For circuit interface channel 0, bits 10-13 contain the 4-bit data field and bits 8 and 9 are always written with zeros. For circuit interface channel 1, bits 8-13 contain the 6-bit data field.

- TEMP1–4—These bits are used internally by the RISC microcontroller.

16.11.8.3 SMC GCI COMMANDS. The following commands are issued to the CPM command register.

- **INIT TX AND RX PARAMS**—This command initializes the transmit and receive parameters in the parameter RAM to their reset state and it is especially useful when switching protocols on a given serial channel.
- **TRANSMIT ABORT REQUEST**—This receiver command can be issued when the MPC823 implements the monitor channel protocol. When it is issued, the MPC823 sends an abort request on the A bit.
- **TIMEOUT**—This transmitter command can be issued when the MPC823 implements the monitor channel protocol and it is usually issued because the device is not responding or A bit errors are detected. The MPC823 sends an abort request on the E bit at the time this command is issued.

16.11.8.4 SMC GCI MODE REGISTER. When the SMC is in GCI mode, the 16-bit, memory-mapped, read/write SMC mode register is referred to as the SMC GCI mode register (SMCMR–GCI). The functions of bits 8–15 are common to each SMC protocol, but bits 0–7 vary according to the protocol selected by the SM bits.

SMCMR–GCI

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RES	CLEN				ME	RES	C#	RES		SM		DM		TEN	REN
RESET	0	0				0	0	0	0		0		0		0	0
R/W	R/W	R/W				R/W	R/W	R/W	R/W		R/W		R/W		R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA82 (SMCMR1), 0xA92 (SMCMR2)															

Bits 0, 6, and 8–9—Reserved

These bits are reserved and should be set to 0.

CLEN—Character Length

This field is used to define the total number of bits in the circuit interface and monitor channels of the SCIT channels 0 or 1. CLEN ranges from 0 to 15 and specifies values from 1 to 16 bits. CLEN should be written with 13 for the SCIT channel 0 or GCI (8 data bits, plus A and E bits, plus 4 circuit interface bits = 14 bits) or with 15 for the SCIT channel 1 (8 data, bits, plus A and E bits, plus 6 circuit interface bits = 16 bits).

ME—Monitor Enable

0 = A serial management controller does not support the monitor channel.
1 = A serial management controller supports the monitor channel with the transparent or monitor channel protocol as defined in the MP bit.

C#—SCIT Channel Number

0 = SCIT channel 0.
1 = SCIT channel 1. Required for Siemens ARCOFI and SGS S/T chips.

SM—SMC Mode

00 = GCI or SCIT support. Required for SMC GCI or SCIT operation.
01 = Reserved.
10 = UART.
11 = Totally transparent operation.

DM—Diagnostic Mode

00 = Normal operation.
01 = Local loopback mode.
10 = Echo mode.
11 = Reserved.

TEN—SMC Transmit Enable

0 = SMC transmitter disabled.
1 = SMC transmitter enabled.

REN—SMC Receive Enable

0 = SMC receiver disabled.
1 = SMC receiver enabled.

16.11.8.5 SMC GCI EVENT REGISTER. When the SMC is in GCI mode, the 8-bit memory-mapped SMC event register is referred to as the SMC GCI event (SMCE–GCI) register. It is used to generate interrupts and report events recognized by the SMC channel. When an event is recognized, a serial management controller sets the corresponding bit in this register. Interrupts generated by this register can be masked in the SMCM–GCI register. A bit is cleared by writing a 1 (writing a zero has no effect) more than one bit can be cleared at a time. All unmasked bits must be cleared before the communication processor module clears the internal interrupt request to the CPM interrupt controller. This register is cleared by reset and can be read at any time.

SMCE–GCI

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED				CTXB	CRXB	MTXB	MRXB
RESET	0				0	0	0	0
R/W	R/W				R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA86 (SMC1), 0xA96 (SMC2)							

Bits 0–3—Reserved

These bits are reserved and should be set to 0.

CTXB—Circuit Interface Channel Buffer Transmitted

This bit indicates that the circuit interface transmit buffer is now empty.

CRXB—Circuit Interface Channel Buffer Received

This bit indicates when the circuit interface receive buffer is full.

MTXB—Monitor Channel Buffer Transmitted

This bit indicates that the monitor transmit buffer is now empty.

MRXB—Monitor Channel Buffer Received

This bit indicates when the monitor receive buffer is full.

16.11.8.6 SMC GCI MASK REGISTER. Whe the SMC is in GCI mode, the 8-bit, memory-mapped, read/write SMC mask register is referred to as the SMC GCI mask (SMCM–GCI) register. It has the same bit format as the SMCE–GCI register. If a bit in this register is a 1, the corresponding interrupt in the SMCE–GCI is enabled. If the bit is zero, the corresponding interrupt in the SMCE–GCI is masked.

SMCM–GCI

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED				CTXB	CRXB	MTXB	MRXB
RESET	0				0	0	0	0
R/W	R/W				R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA8A (SMC1), 0xA9A(SMC2)							

16.12 THE SERIAL PERIPHERAL INTERFACE

The serial peripheral interface (SPI) allows the MPC823 to exchange data between other MPC8xx Family chips, the MC68328, MC68360, and MC68302 embedded microprocessors, as well as the MC68HC11 and MC68HC05 microcontroller families and a variety of peripheral devices.

The serial peripheral interface is a full-duplex, synchronous, character-oriented channel that supports a four-wire interface (receive, transmit, clock and slave select). The SPI block consists of transmitter and receiver sections, an independent baud rate generator, and a control unit. The transmitter and receiver sections use the same clock, which is derived from the SPI baud rate generator in master mode and generated externally in slave mode. During an SPI transfer, data is transmitted and received simultaneously.

Becuase the SPI receiver and transmitter are double-buffered, as illustrated in the block diagram below, the effective FIFO size is 2 characters. You can program the MPC823 serial peripheral interface to shift out the most- or least-significant bit first. When the serial peripheral interface is not enabled in the SPMODE register, it consumes very little power.

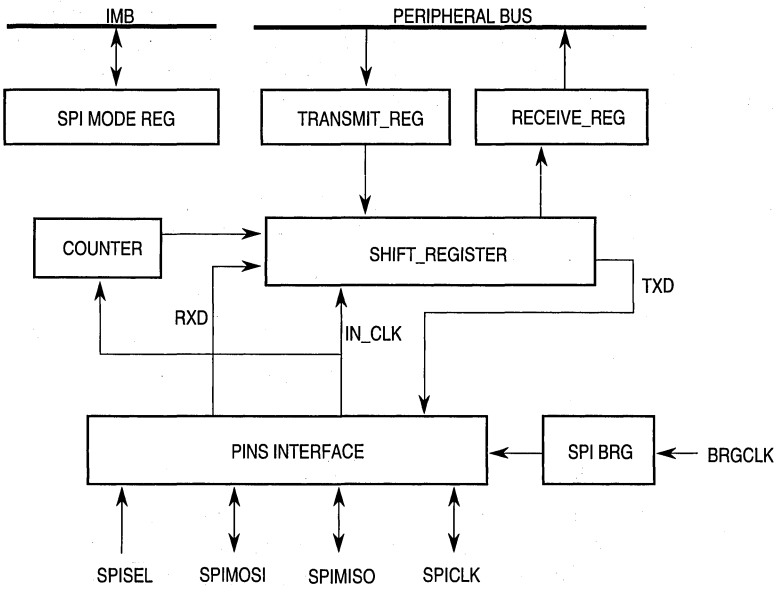


Figure 16-120. SPI Block Diagram

16.12.1 Features

The following is a list of the serial peripheral interface's main features:

- Four-Wire Interface (SPIMOSI, SPIMISO, SPICLK, and $\overline{\text{SPISEL}}$)
- Full-Duplex Operation
- Works with Data Characters from 4 to 16 bits Long
- Supports Back-to-Back Character Transmission and Reception
- Master or Slave SPI Modes Supported
- Multimaster Environment Support
- Continuous Transfer Mode for Autoscanning Peripherals
- Supports maximum Clock Rates of 6.25MHz in Master Mode and 12.5MHz in Slave Mode, Assuming a 25MHz System Clock is Used
- Independent Programmable Baud Rate Generator
- Programmable Clock Phase and Polarity
- Open-Drain Output Pins Support Multimaster Configuration
- Local Loopback Capability for Testing

16.12.2 SPI Clocking and Pin Functions

You can configure the serial peripheral interface as a master for the serial channel or as a slave. You can also use it in a multimaster environment. When the serial peripheral interface is a master, use the SPI baud rate generator to generate the SPI transmit and receive clocks. It takes its input from the BRGCLK, which is generated in the clock synthesizer of the MPC823, specifically for the SPI baud rate generator and the other three baud rate generators in the communication processor module.

The SPIMISO pin is an input in master mode and an output in slave mode. It follows then, that the SPIMOSI pin is an output in master mode and an input in slave mode. The reason the pin names SPIMOSI and SPIMISO change functionality between master and slave mode is to support a multimaster configuration that allows communication from one serial peripheral interface to another with the same hardware configuration.

When the serial peripheral interface is in master mode, SPICLK is the clock output signal that shifts in the received data from the SPIMISO pin and shifts out the transmitted data to the SPIMOSI pin. Additionally, an SPI master device must provide a slave select signal output to enable the SPI slave devices. You can implement this by using one of the MPC823 general-purpose I/O pins. The $\overline{\text{SPISEL}}$ pin should not be asserted while the serial peripheral interface is in master mode or an error will occur.

When the serial peripheral interface is in slave mode, SPICLK is the clock input signal that shifts in the received data from the SPIMOSI pin and shifts out the transmitted data to the SPIMISO pin. The $\overline{\text{SPISEL}}$ pin provided by the MPC823 is the enable input to the SPI slave. When the serial peripheral interface is operating in a multimaster environment, the $\overline{\text{SPISEL}}$ pin is still an input and is used to detect an error condition when more then one master is operating.

Using the fields in the SPI mode register, you can select any of the four combinations of the gated SPICLK phase and polarity. The SPI pins can also be configured as open-drain pins to support a multimaster configuration in which the same SPI pin is driven by the MPC823 or an external SPI device.

16.12.3 The SPI Transmission and Reception Process

When the serial peripheral interface is in master mode, it transmits a message to the peripheral or slave, which sends back an immediate reply. When the MPC823 has more than one slave, it can use the general-purpose parallel I/O pins to selectively enable different slaves. To start the data exchange process, the core writes the data to be transmitted into a data buffer, configures a TX buffer descriptor with its R bit set, and configures one or more RX buffer descriptors. The core then sets the STR bit in the SPCOM register to start transmitting data, which starts when the SDMA channel loads the transmit FIFO with data.

The serial peripheral interface then generates programmable clock pulses on the SPICLK pin for each character and shifts the data out on the SPIMOSI pin. At the same time, the serial peripheral interface shifts received data in from the SPIMISO pin. This received data is written into a receive buffer using the next available RX buffer descriptor. The serial peripheral interface continues transmitting and receiving characters until the transmit buffer has been completely transmitted or an error has occurred. The communication processor module then clears the R and E bits in the TX buffer descriptor and RX buffer descriptor and may issue a maskable interrupt to the CPM interrupt controller.

When multiple TX buffer descriptors are ready to be transmitted, the TX buffer descriptor L bit determines whether or not the serial peripheral interface should continue transmitting without waiting for the STR bit to be set again. If the L bit is cleared, the data from the next TX buffer descriptor begins transmitting after data from the first TX buffer descriptor is transmitted. If the L bit is set, transmission stops after data from this TX buffer descriptor has finished transmitting. In addition, the current RX buffer descriptor that is used to receive data is closed after transmission stops, even if the receive buffer is not full. This means that you do not need to provide receive buffers that are the same length as the transmit buffers. If the serial peripheral interface is the only master in a system, then the $\overline{\text{SPISEL}}$ pin can be used as a general-purpose I/O, and the internal $\overline{\text{SPISEL}}$ signal to the serial peripheral interface is always forced internally inactive, thus eliminating the possibility of a multimaster error.

When the serial peripheral interface is in slave mode, it receives messages from an SPI master and sends back a simultaneous reply. The $\overline{\text{SPISEL}}$ pin must be asserted before receive clocks are recognized and once $\overline{\text{SPISEL}}$ is asserted, the SPICLK pin becomes an input from the master to the slave. SPICLK can be any frequency from the DC to the BRGCLK/2, which is 12.5MHz for a 25MHz system.

Before the data is exchanged, the core writes the data to be transmitted into a data buffer, configures a TX buffer descriptor with its R bit set, and configures one or more RX buffer descriptors. The core then sets the STR bit in the SPCOM register to enable the serial peripheral interface so it will prepare the data for transmission and wait for the $\overline{\text{SPISEL}}$ pin to be asserted. Data is shifted out from the slave on the SPIMISO pin and shifted in through the SPIMOSI pin. A maskable interrupt is issued when a full buffer finishes receiving and transmitting or after an error occurs. Using the next RX buffer descriptor in the ring, the serial peripheral interface continues reception until it runs out of receive buffers or the $\overline{\text{SPISEL}}$ pin is negated.

Transmission continues until no more data is available or the SPISEL pin is negated. If the pin is negated before all the data is transmitted, it stops, but the TX buffer descriptor stays open. Further transmission continues once the SPISEL pin is reasserted and SPICLK begins toggling. After the characters in the TX buffer descriptor are transmitted, the serial peripheral interface transmits ones if SPISEL is not negated.

16.12.3.1 MULTIMASTER OPERATION. The serial peripheral interface can operate in a multimaster environment in which some SPI devices are connected to the same bus. In this configuration, the SPIMOSI, SPIMISO, and SPICLK pins of all SPIs are connected together and the SPISEL input pins are connected separately. In this environment, only one SPI device can be in master mode and all the others must be in slave mode. When the serial peripheral interface is configured as a master and its SPISEL signal goes active or low, a multimaster error will occur because more than one SPI device is a bus master. The serial peripheral interface sets the MME bit in the SPIE register and a maskable interrupt is issued to the core. It also disables SPI operation and the output drivers of the SPI pins. The core should clear the EN bit the SPMODE register before using the serial peripheral interface again. After the problems are corrected, clear the MME bit and enable the serial peripheral interface the same way you would after a reset.

Note: The maximum sustained data rate that the serial peripheral interface supports is SYSTEMCLK/50. However, the serial peripheral interface can transfer a single character at much higher rates. For instance, SYSTEMCLK/4 in master mode and SYSTEMCLK/2 in slave mode. If multiple characters are to be transmitted, you should insert gaps between them so that it will not exceed the maximum sustained data rate.

16.12.3.2 SPI PARAMETER RAM MEMORY MAP. The SPI parameter RAM area begins at the SPI base address and is used for the general SPI parameters. Notice that it is similar to the SCC2 general-purpose parameter RAM. You must initialize certain parameter RAM values before the serial peripheral interface is enabled. The communication processor module initilizes the other values. Once initialized, the parameter RAM values do not usually need to be accessed by your software. They should only be modified when there is no serial peripheral interface activity in progress.

Table 16-39. SPI Parameter RAM Memory Map

ADDRESS	NAME	WIDTH	DESCRIPTION
SPI Base + 00	**RBASE**	Half-word	RX BD Base Address
SPI Base+ 02	**TBASE**	Half-word	TX BD Base Address
SPI Base+ 04	**RFCR**	Byte	RX Function Code
SPI Base+ 05	**TFCR**	Byte	TX Function Code
SPI Base+ 06	**MRBLR**	Half-word	Maximum Receive Buffer Length
SPI Base+ 08	RSTATE	Word	RX Internal State
SPI Base+ 0C	RPTR	Word	RX Internal Data Pointer
SPI Base + 10	RBPTR	Half-word	RX BD Pointer
SPI Base + 12	RCNT	Half-word	RX Internal Byte Count
SPI Base + 14	RTMP	Word	RX Temp
SPI Base + 18	TSTATE	Word	TX Internal State
SPI Base + 1C	TPTR	Word	TX Internal Data Pointer
SPI Base + 20	TBPTR	Half-word	TX BD Pointer
SPI Base + 22	TCNT	Half-word	TX Internal Byte Count
SPI Base + 24	TTMP	Word	TX Temp

NOTE: You are only responsible for initializing the items in bold.
SPI Base = (IMMR & 0xFFFF0000) + 0x3D80.
SCC2 Ethernet parameter RAM space overlaps the SPI parameter RAM space. The address range for SCC2 space is 0x1000 through 0x1CA3. You need a microcode patch to run SPI and Ethernet concurrently.

- RBASE and TBASE—The receive and transmit buffer descriptor base address entries are where the dual-port RAM starts receiving and transmitting data for the RX and TX buffer descriptors. They provide a great deal of flexibility in how buffer descriptors for a serial peripheral interface are partitioned. You must initialize these entries before enabling the corresponding channel. You should not configure the SPI buffer descriptor tables to overlap with the tables of the USB, SMC, and SCC or erratic operation will occur. RBASE and TBASE should contain a value that is divisible by eight.

- RFCR and TFCR—Threceive and transmit function code register entries contain the value that you want to appear on the AT pins when the associated SDMA channel accesses memory. This register controls the byte-ordering convention used in the transfers.

RFCR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED			BO		AT		
RESET	0			0		0		
R/W	R/W			R/W		R/W		
ADDR	SPI BASE + 0x04							

Bits 0–2—Reserved

These bits are reserved and should be set to 0.

BO—Byte Ordering

You should set these bits to select the required byte ordering of the data buffer.

00 = The DEC/Intel convention is used for byte ordering (swapped operation) and is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation) is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT—Address Type 1–3

These bits contain the function code value used during the SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA-type access.

TFCR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED			BO		AT		
RESET	0			0		0		
R/W	R/W			R/W		R/W		
ADDR	SPI BASE + 0x05							

Bits 0–2—Reserved

These bits are reserved and should be set to 0.

BO—Byte Ordering

You should set these bits to select the required byte ordering of the data buffer. If this bit field is modified on-the-fly, it takes effect at the beginning of the next frame or at the beginning of the next buffer descriptor.

00 = The DEC/Intel convention is used for byte ordering (swapped operation) and is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation) is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT—Address Type 1–3

These bits contain the function code value used during the SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA-type access.

- MRBLR—The serial peripheral interface has one maximum receive buffer length entry to define its receive buffer length and it defines the maximum number of bytes that the MPC823 writes to a receive buffer on the serial peripheral interface before moving to the next buffer. The MPC823 can write fewer bytes to the buffer than the MRBLR value if an error or end-of-frame occurs, but it never writes more bytes than the MRBLR value. Buffers you supply for the MPC823 to use should always be at least as long as MRBLR.

  The transmit buffers for a serial peripheral interface are not affected by the value you program into MRBLR and they can be individually chosen to have varying lengths, as needed. You choose the number of bytes to be transmitted by programming the DATA LENGTH field in the TX buffer descriptor.

  MRBLR is not intended to be dynamically changed while a serial peripheral interface is operating. However, if it is modified in a single bus cycle with one 16-bit move (not two 8-bit bus cycles back-to-back), a dynamic change in receive buffer length can be successfully achieved. This occurs when the communication processor module moves control to the next RX buffer descriptor in the table. Thus, a change to MRBLR does not have an immediate effect. To guarantee the exact RX buffer descriptor on which the change occurs, you should only change MRBLR while the SPI receiver is disabled. The MRBLR value should be greater than zero and should be even if the character length of the data is greater than 8 bits.

- RRBPTR—The RX buffer descriptor pointer entry for each SPI channel points to the next buffer descriptor that the receiver transfers data to when it is in an idle state or to the current buffer descriptor during frame processing. After a reset or when the end of the buffer descriptor table is reached, the communication processor module initializes this pointer to the value programmed in the RBASE entry. Although in most applications you should not write RBPTR, it can be modified when the receiver is disabled or when you are sure that no receive buffer is currently in use.
- TBPTR—The TX buffer descriptor pointer entry for each SPI channel points to the next buffer descriptor that the transmitter transfers data from when it is in an idle state or to the current buffer descriptor during frame transmission. After a reset or when the end of buffer descriptor table is reached, the communication processor module initializes this pointer to the value programmed in the TBASE entry. Although in most applications you should not write TBPTR, it can be modified when the transmitter is disabled or when you are sure that no transmit buffer is currently in use.
- Other General Parameters—For normal operation, you do not need to access these parameters. They are only listed here because they provide helpful information for experienced users and they can be used for debugging purposes. Additional parameters are listed in Table 16-39. RPTR and TPTR are updated by the SDMA channels to show the next address in the buffer to be accessed. TCNT is a down-count value initialized with the TX buffer descriptor data length and decremented with every byte read by the SDMA channels. The RCNT is a down-count value that is initialized with the MRBLR value and decremented with every byte the SDMA channels write. The RSTATE, TSTATE, RTMP, TMP, and reserved areas can only be used by the RISC microcontroller.

Note: To extract data from a partially full buffer, use the **CLOSE RX BD** command.

16.12.3.3 SPI COMMANDS. The following transmit and receive commands are issued to the CPM command register (CPCR).

- **INIT TX PARAMETERS**—This command initializes all transmit parameters in this serial channel parameter RAM to their reset state and should only be issued when the transmitter is disabled. The **INIT TX AND RX PARAMS** command can also be used to reset the transmit and receive parameters.
- **CLOSE RX BD**—This command is used to force the SPI controller to close the current RX buffer descriptor if it is currently being used and to use the next buffer descriptor for any subsequently received data. If the SPI controller is not in the process of receiving data, no action is taken by this command.
- **INIT RX PARAMETERS**—This command initializes all the receive parameters in this serial channel parameter RAM to their reset state and should only be issued when the receiver is disabled. The **INIT TX AND RX PARAMS** command can also be used to reset the receive and transmit parameters.

16.12.3.4 SPI BUFFER DESCRIPTOR RING. The data associated with the serial peripheral interface is stored in buffers, which are referenced by buffer descriptors organized in a buffer descriptor ring located in the dual-port RAM. This ring has the same basic configuration as the SCC2, SMC, USB, and I^2C controllers.

The buffer descriptor ring forms a circular queue that helps you arrange the buffers you want to transmit or receive. Using the buffer descriptors, the communication processor module confirms reception and transmission or indicates error conditions so that the processor knows the buffers have been serviced. The actual buffers can reside in either external memory or internal memory and the data buffers can reside in the parameter area of another unused controller if it is not enabled.

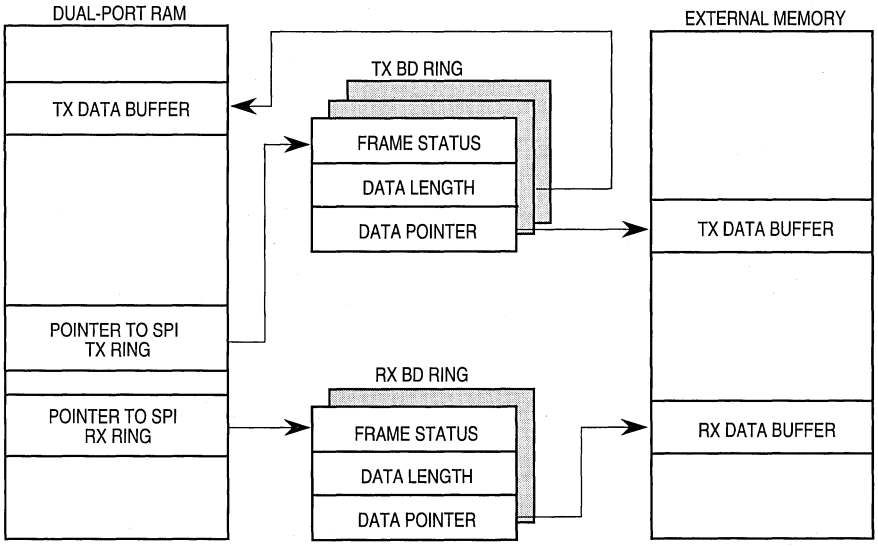


Figure 16-121. SPI Memory Format

16.12.4 Programming the Serial Peripheral Interface

16.12.4.1 SPI MODE REGISTER. The read/write SPI mode (SPMODE) register controls both the serial peripheral interface operation mode and clock source. Table 16-2 contains more information on commands that can be used with this register.

SPMODE

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RES	LOOP	CI	CP	DIV16	REV	M/S	EN	LEN				PM			
RESET	0	0	0	0	0	0	0	0	0				0			
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W				R/W			
ADDR	(IMMR & 0xFFFF0000) + 0xAA0															

Bit 0—Reserved

This bit is reserved and should be set to 0.

LOOP—Loop Mode

When set, this bit selects the local loopback operation. The transmitter output is internally connected to the receiver input. The receiver and transmitter operate normally, except that the externally received data is ignored.

0 = Normal operation.
1 = The serial peripheral interface is in loopback mode.

CI—Clock Invert

This bit inverts the SPI clock polarity. See Figure 16-122 and Figure 16-123 for details.

0 = The inactive state of SPICLK is low.
1 = The inactive state of SPICLK is high.

CP—Clock Phase

This bit selects one of two fundamentally different transfer formats. See Figure 16-122 and Figure 16-123 for details.

0 = SPICLK starts toggling at the middle of the data transfer.
1 = SPICLK starts toggling at the beginning of the data transfer.

DIV16—Divide by 16

This bit selects the clock source for the SPI baud rate generator when configured as an SPI master. In slave mode, the clock source is the SPICLK pin.

0 = Use the BRGCLK as the input to the SPI baud rate generator.
1 = Use the BRGCLK/16 as the input to the SPI baud rate generator.

REV—Reverse Data

This bit determines the receive and transmit character bit order.

0 = Reverse data. Least-significant bit of the character transmitted and received first.
1 = Normal operation. Most-significant bit of the character transmitted and received first.

M/S—Master/Slave

This bit configures the serial peripheral interface to operate as a master or slave.

0 = The serial peripheral interface is a slave.
1 = The serial peripheral interface is a master.

EN—Enable SPI

This bit enables serial peripheral interface operation. Configure the SPIMOSI, SPIMISO, SPICLK, and $\overline{\text{SPISEL}}$ to connect to the serial peripheral interface as described in **Section 16.14.6 The Port B Registers**. When the EN bit is cleared, the serial peripheral interface is in a reset state and consumes minimal power, which means the SPI baud rate generator is not functioning and the input clock is disabled.

0 = The serial peripheral interface is disabled.
1 = The serial peripheral interface is enabled.

Note: You should not modify other bits of the SPMODE register when the EN bit is set.

LEN—Character Length

This field specifies the number of bits in a character. These values must be between 4 and 16 bits. If you program a value less than 4 bits, erratic behavior will occur. If the value of LEN is less than or equal to a byte, there will be LEN number of valid bits in every byte (8 bits) in memory. On the other hand, if the value of LEN is greater than a byte, there is LEN number of valid bits in every half-word (16 bits) in memory.

0011 = 4-bit character length.
0100 = 5-bit character length.
0101 = 6-bit character length.
0110 = 7-bit character length.
0111 = 8-bit character length.
•
•
•
1111 = 16-bit character length.

PM—Prescale Modulus Select

This field specifies the divide ratio of the prescale divider in the SPI clock generator. The BRGCLK is divided by 4 * ([PM0–PM3] + 1), thus giving a clock divide ratio of 4 to 64. The clock has a 50% duty cycle.

16.12.4.1.1 SPI Examples With Different LEN Values. The programming examples below illustrate the effect of the LEN field and the REV bit in the SPMODE register on output from the SPI controller. They illustrate the master mode output from the SPI controller as the LEN varies. To help map the output process, make *g* through *v* the binary symbols, use *x* to indicate a deleted bit, use __ to indicate original byte boundaries, and use _ to indicate original nibble (4-bit) boundaries.

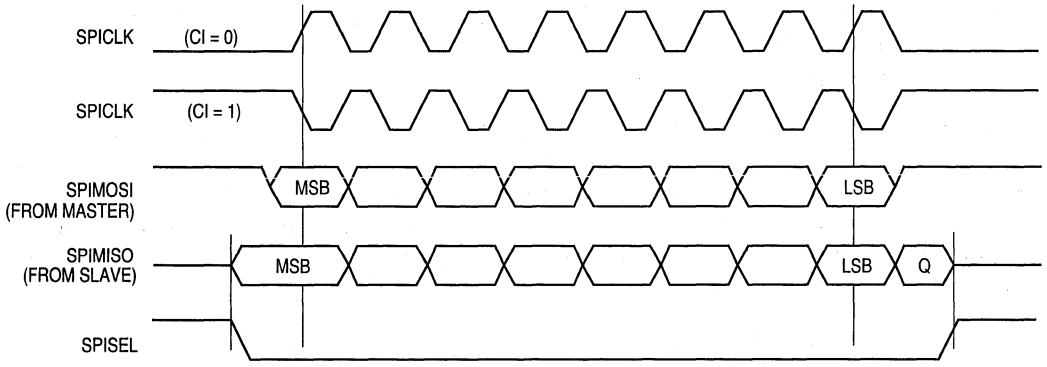
The initial pattern for all examples is `ghij_klmn__opqr_stuv`.

Example 1 `LEN = 0x4` `(Data Size = 5)`
Data Selected: `xxxj_klmn_xxxr_stuv`
Data Transmitted for REV=0: `nmlk_j__vuts_r`
Data Transmitted for REV=1: `j_nmlk__r_stuv`

Example 2 `LEN = 0x7` `(Data Size = 8)`
Data Selected: `ghij_klmn_opqr_stuv`
Data Transmitted for REV=0: `nmlk_jihg__vuts_rqpo`
Data Transmitted for REV=1: `ghij_klmn__opqr_stuv`

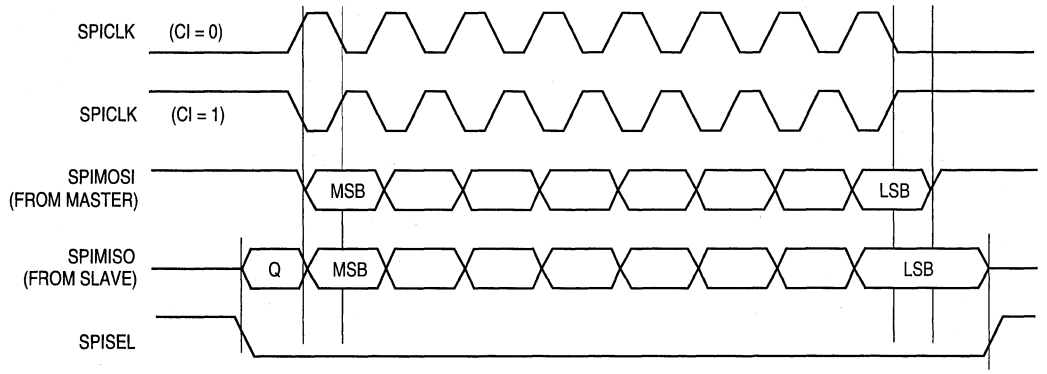
Example 3 `LEN = 0xc` `(Data Size = 13)`
Data Selected: `ghij_klmn_xxxr_stuv`
Data Transmitted for REV=0: `nmlk_jihg__vuts_r`
Data Transmitted for REV=1: `r_stuv__ghij_klmn`

Example 4 `LEN = 0xf` `(Data Size = 16)`
Data Selected: `ghij_klmn_opqr_stuv`
Data Transmitted for REV=0: `nmlk_jihg__vuts_rqpo`
Data Transmitted for REV=1: `opqr_stuv__ghij_klmn`



NOTE: Q = Undefined Signal

Figure 16-122. SPI Transfer Format If CP = 0



NOTE: Q = Undefined Signal

Figure 16-123. SPI Transfer Format If CP = 1

16.12.4.1.2 SPI Receive Buffer Descriptor. Using receive (RX) buffer descriptors, the communication processor module reports information about each buffer of received data, closes the current buffer, generates a maskable interrupt, and starts receiving data in the next buffer once the current buffer is full. Additionally, it closes the buffer when the serial peripheral interface is configured as a slave and the $\overline{\text{SPISEL}}$ pin goes inactive, thus indicating that the reception process has stopped.

The first word of the RX buffer descriptor contains status and control bits that you prepare before reception and then the communication processor module sets them after the buffer is closed. The second word contains the data length (in bytes) that is received and the third and fourth words contain a pointer that always points to the beginning of the received data buffer. You should configure the RX buffer descriptor bits before the serial peripheral interface is enabled.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	**E**	RES	**W**	**I**	L	RES	**CM**	RESERVED							OV	ME
OFFSET + 2	DATA LENGTH															
OFFSET + 4	**RX DATA BUFFER POINTER**															
OFFSET + 6																

NOTE: You are only responsible for initializing the items in bold.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

E—Empty

0 = The data buffer associated with this RX buffer descriptor is filled with data or stops receiving data because an error occurred. The core is free to examine or write to any fields of this RX buffer descriptor, but it does not use this buffer descriptor as long as the E bit is zero.

1 = The data buffer associated with this buffer descriptor is empty or is currently receiving data. This RX buffer descriptor and its associated receive buffer are owned by the communication processor module. Once the E bit is set, the core should not write any fields of this RX buffer descriptor.

Bit 1, 5, and 7–13—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the RX buffer descriptor table.
1 = This is the last buffer descriptor in the RX buffer descriptor table. After this buffer is used, the communication processor module receives incoming data into the first buffer descriptor that RBASE points to in the table. The number of RX buffer descriptors in this table is programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0 No interrupt is generated after this buffer is filled.
1 The RXB bit in the SPIE register is set when this buffer is completely filled by the communication processor module, indicating the need for the core to process the buffer. The RXB bit can cause an interrupt if it is enabled.

CM—Continuous Mode

This bit is valid only when the serial peripheral interface is in master mode. In slave mode, it should be written as a zero.

0 = Normal operation.
1 = The E bit is not cleared by the communication processor module after this buffer descriptor is closed, thus allowing the associated data buffer to be automatically overwritten next time the communication processor module accesses this buffer descriptor. This allows continuous reception from an SPI slave into one buffer for autoscanning of a serial A/D peripheral with no core overhead.

L—Last

This bit is set by the serial peripheral interface when the buffer is closed because the $\overline{\text{SPISEL}}$ pin was negated. This only occurs when the serial peripheral interface is in slave mode. Otherwise, the ME bit is set. The serial peripheral interface writes this bit after the received data is placed into the associated data buffer.

0 = This buffer does not contain the last character of the message.
1 = This buffer contains the last character of the message.

OV—Overrun

This bit indicates that a receiver overrun has occurred during reception. This can only occur when the serial peripheral interface is in slave mode. The serial peripheral interface writes this bit after the received data is placed into the associated data buffer.

ME—Multimaster Error

This bit indicates that this buffer is closed because the $\overline{\text{SPISEL}}$ pin was asserted when the serial peripheral interface was in master mode. This indicates a synchronization problem between multiple masters on the SPI bus. The serial peripheral interface writes this bit after the received data is placed into the associated data buffer.

DATA LENGTH

This field represents the number of octets that the communication processor module writes into this buffer descriptor data buffer. The communication processor module writes it once as the buffer descriptor is closed. The serial peripheral interface writes these bits after the received data is placed into the associated data buffer. The actual amount of memory allocated for this buffer should be greater than or equal to the MRBLR.

RX DATA BUFFER POINTER

This field always points to the first location of the associated data buffer, must be even, and can reside in internal or external memory. The serial peripheral interface writes these bits after the received data is placed into the associated data buffer.

16.12.4.1.3 SPI Transmit Buffer Descriptor. Data to be transmitted with the serial peripheral interface is sent to the communication processor module by arranging it in buffers referenced by the transmit (TX) buffer descriptor ring. The first word of the TX buffer descriptor contains status and control bits. You should prepare the following bits before transmitting data.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	R	RES	W	I	L	RES	CM	RESERVED							UN	ME
OFFSET + 2	**DATA LENGTH**															
OFFSET + 4	**TX DATA BUFFER POINTER**															
OFFSET + 6																

NOTE: You are only responsible for initializing the items in bold.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

R—Ready

0 = The data buffer associated with this buffer descriptor is not ready for transmission and you are free to manipulate this buffer descriptor or its associated data buffer. The communication processor module clears this bit after the buffer is transmitted or after an error occurs.

1 = The data buffer, which you prepare for transmission, is not transmitted yet or is currently being transmitted. You cannot write any fields of this buffer descriptor once this bit is set.

Bits 1, 5, and 7–13—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the TX buffer descriptor table.
1 = This is the last buffer descriptor in the TX buffer descriptor table. After this buffer is used, the communication processor module receives incoming data into the first buffer descriptor that TBASE points to in the table. The number of TX buffer descriptors in this table is programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is serviced.
1 = The TXB or TXE bit in the event register is set when this buffer is serviced. TXB and TXE can cause interrupts if they are enabled.

L—Last

0 = This buffer does not contain the last character of the message.
1 = This buffer contains the last character of the message.

CM—Continuous Mode

This bit is only valid when the serial peripheral interface is in master mode. In slave mode, it should be written as a zero.

0 = Normal operation.
1 = The R bit is not cleared by the communication processor module after this buffer descriptor is closed, thus allowing the associated data buffer to be automatically retransmitted next time the communication processor module accesses this buffer descriptor.

UN—Underrun

This bit indicates that the serial peripheral interface has encountered a transmitter underrun condition while transmitting the associated data buffer. This error condition is only valid when the serial peripheral interface is in slave mode. The serial peripheral interface writes this bit after it finishes transmitting the associated data buffer.

ME—Multimaster Error

This bit indicates that this buffer is closed because the $\overline{\text{SPISEL}}$ pin was asserted when the serial peripheral interface was in master mode. This indicates a synchronization problem between multiple masters on the SPI bus. The serial peripheral interface writes this bit after it finishes transmitting the associated data buffer.

DATA LENGTH

This field indicates the number of octets that the communication processor module should transmit from this buffer descriptor data buffer. However, it is never modified by the communication processor module. Normally, this value should be greater than zero, but if the number of data bits in the character is greater than 8, then the data length should be even. For example, to transmit three characters of 8-bit data, 1 start, and 1 stop, the data length field should be initialized to 3. However, to transmit three characters of 9-bit data, the DATA LENGTH field should be initialized to 6 since the three 9-bit data fields occupy three half-words in memory. The serial peripheral interface writes these bits after it finishes transmitting the associated data buffer.

TX DATA BUFFER POINTER

This field always points to the first location of the associated data buffer. They can be even or odd, unless the number of actual data bits in the character is greater than 8 bits, in which case the transmit buffer pointer must be even. The buffer can reside in internal or external memory. The serial peripheral interface writes these bits after it finishes transmitting the associated data buffer.

16.12.4.2 SPI COMMAND REGISTER. The 8-bit read/write SPI command (SPCOM) register is used to start serial peripheral interface operation.

SPCOM

BIT	0	1	2	3	4	5	6	7
FIELD	STR	RESERVED						
RESET	0	0						
R/W	R/W	R/W						
ADDR	(IMMR & 0xFFFF0000) + 0xAAD							

STR—Start Transmit

When the serial peripheral interface is configured as a master, setting this bit to 1 causes the serial peripheral interface to start transmitting and receiving data to and from the transmit/receive buffers if they are ready. When the serial peripheral interface is in slave mode, setting the STR bit to 1 when the serial peripheral interface is idle causes the serial peripheral interface to load the transmit data register from the SPI transmit buffer and start transmission as soon as the next SPI input clocks and select signal are received. This bit is automatically cleared after one system clock cycle.

Bits 1–7—Reserved.

These bits are reserved and should be set to 0.

16.12.4.3 SPI EVENT REGISTER. The 8-bit memory-mapped SPI event (SPIE) register is used to generate interrupts and report events recognized by the serial peripheral interface. When an event is recognized, the serial peripheral interface sets its corresponding bit in this register. Interrupts generated by this register can be masked in the SPIM register. A bit is cleared by writing a 1 (writing a zero has no effect) and more than one bit can be cleared at a time. However, all unmasked bits must be cleared before the communication processor module clears the internal interrupt request. This register is cleared by reset and can be read at any time.

SPIE

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED		MME	TXE	RES	BSY	TXB	RXB
RESET	0		0	0	0	0	0	0
R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xAA6							

Bits 0–1 and 4—Reserved

These bits are reserved and should be set to 0.

MME—Multi-Master Error

This bit indicates that the serial peripheral interface has found out that the $\overline{\text{SPISEL}}$ pin was asserted externally while the serial peripheral interface was in master mode.

TXE—TX Error

This bit indicates that an error has occurred during transmission.

BSY—Busy Condition

This bit indicates that received data has been discarded due to a lack of buffers. This bit is set after the first character is received for which there is no receive buffer available.

TXB—TX Buffer

This bit indicates that a buffer has been transmitted. It is set once the transmit data of the last character in the buffer is written to the transmit FIFO. You must wait two character times to be sure that the data is completely sent over the transmit pin.

RXB—RX Buffer

This bit indicates that a buffer has been received. It set after the last character is written to the receive buffer and the RX buffer descriptor is closed.

16.12.4.4 SPI MASK REGISTER. The 8-bit read/write SPI mask (SPIM) register has the same bit formats as the SPIE register. If a bit in the SPIM is 1, the corresponding interrupt in the SPIE register is enabled. If the bit is zero, the corresponding interrupt in the SPIE register is masked. This register is cleared by reset.

SPIM

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED		MIME	TXE	RES	BSY	TXB	RXB
RESET	0		0	0	0	0	0	0
R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xAAA							

16.12.5 SPI Master Programming Example

The following is an example initialization sequence for using the peripheral interface in master mode at a high speed.

1. Configure the port B pins to enable the SPIMOSI, SPIMISO, and SPICLK pins. Write PBPAR and PBDIR bits 30, 29, and 28 with ones and then PBODR bits 30, 29, and 28 with zeros.

Note: For multimaster operation, enable the $\overline{\text{SPISEL}}$ pin to internally connect to the serial peripheral interface.

2. Configure a parallel I/O pin to operate as the serial peripheral interface select pin if needed. If PB16 is chosen, write PBODR bit 16 with a zero, PBDIR bit 16 with a one, and PBPAR bit 16 with a zero. Then write PBDAT bit 16 with a zero to constantly assert the select pin.
3. Write RBASE and TBASE in the SPI parameter RAM to point to the RX buffer descriptor and TX buffer descriptor in the dual-port RAM. Assuming one RX buffer descriptor at the beginning of the dual-port RAM and one TX buffer descriptor following that RX buffer descriptor, write RBASE with 0x2000 and TBASE with 0x2008.
4. Program the CPM command register (CPCR) to execute the **INIT RX AND TX PARAMS** command. Write 0x0051 to the CPCR.
5. Write 0x0001 to the SDCR to initialize the SDMA configuration register.
6. Write 0x18 to RFCR and TFCR for normal operation.
7. Write MRBLR with the maximum number of bytes per receive buffer. For this case, assume 16 bytes, so MRBLR = 0x0010.
8. Initialize the RX buffer descriptor and assume the RX data buffer is at 0x00001000 in main memory. Write 0xB000 to RX_BD_Status, 0x0000 to RX_BD_Length (optional), and 0x00001000 to RX_BD_Pointer.

9. Initialize the TX buffer descriptor and assume the TX data buffer is at 0x00002000 in main memory and contains five 8-bit characters. Write 0xB800 to TX_BD_Status, 0x0005 to TX_BD_Length, and 0x00002000 to TX_BD_Pointer.
10. Write 0xFF to the SPIE register to clear any previous events.
11. Write 0x37 to the SPIM register to enable all possible serial peripheral interface interrupts.
12. Write 0x00000020 to the CIMR to allow the serial peripheral interface to generate a system interrupt. The CICR should also be initialized.
13. Write 0x0370 to the SPMODE register to enable normal operation (not loopback), master mode, serial peripheral interface enabled, 8-bit characters, and the fastest speed possible.
14. Set the STR bit in the SPCOM register to start the transfer.

Note: After 5 bytes are transmitted, the TX buffer descriptor is closed. Additionally, the receive buffer is closed after 5 bytes are received because the L bit of the TX buffer descriptor is set.

16.12.6 SPI Slave Programming Example

The following is an example initialization sequence to follow when the serial peripheral interface is in slave mode. It is very similar to the serial peripheral interface master example, except that the SPISEL pin is used instead of a general-purpose I/O pin.

1. Configure the port B pins to enable the SPIMOSI, SPIMISO, $\overline{\text{SPISEL}}$, and SPICLK pins. Write PBPAR bits 31, 30, 29, and 28 with ones and the PBODR bits 31, 30, 29, and 28 with zeros. Write PBDIR bits 30, 29, and 28 with ones and bit 31 with zero.
2. Write RBASE and TBASE in the SPI parameter RAM to point to the RX buffer descriptor and TX buffer descriptor in the dual-port RAM. Assuming one RX buffer descriptor at the beginning of the dual-port RAM and one TX buffer descriptor following that RX buffer descriptor, write RBASE with 0x2000 and TBASE with 0x2008.
3. Write 0x18 to RFCR and TFCR for normal operation.
4. Program the CPM command register (CPCR) to execute the **INIT RX AND TX PARAMS** command. Write 0x0051 to the CPCR.
5. Write 0x0001 to the SDCR to initialize the SDMA configuration register.
6. Write MRBLR with the maximum number of bytes per receive buffer. Assume 16 bytes, so MRBLR = 0x0010.
7. Initialize the RX buffer descriptor and assume the RX data buffer is at 0x00001000 in main memory. Write 0xB000 to RX_BD_Status, 0x0000 to RX_BD_Length (optional), and 0x00001000 to RX_BD_Pointer.
8. Initialize the TX buffer descriptor and assume the TX data buffer is at 0x00002000 in main memory and contains five 8-bit characters. Write 0xB800 to TX_BD_Status, 0x0005 to TX_BD_Length, and 0x00002000 to TX_BD_Pointer.

9. Write 0xFF to the SPIE register to clear any previous events.
10. Write 0x37 to the SPIM register to enable all possible serial peripheral interface interrupts.
11. Write 0x00000020 to the CIMR to allow the serial peripheral interface to generate a system interrupt. The CICR should also be initialized.
12. Write 0x0170 to the SPMODE register to enable normal operation (not loopback), master mode, serial peripheral interface enabled, and 8-bit characters. The SPI baud rate generator speed is ignored because the serial peripheral interface is in slave mode.
13. Set the STR bit in the SPCOM register to enable the serial peripheral interface to be ready once the master begins the transfer.

Note: If the master transmits 3 bytes and negates the $\overline{\text{SPISEL}}$ pin, the RX buffer descriptor is closed but the TX buffer descriptor remains open. If the master transmits 5 or more bytes, the TX buffer descriptor is closed after the fifth byte. If the master transmits 16 bytes and negates the $\overline{\text{SPISEL}}$ pin, the RX buffer descriptor is closed with no errors and no out-of-buffers error occurs. If the master transmits more than 16 bytes, the RX buffer descriptor is closed (completely full) and the out-of-buffers error occurs after the 17th byte is received.

16.12.7 Handling Interrupts in the SPI

The following sequence should be followed to handle interrupts in the serial peripheral interface.

1. Once an interrupt occurs, read the SPIE register to find out what caused the interrupts. Normally, the SPIE bits should be cleared at this time.
2. Process the TX buffer descriptor to reuse it and the RX buffer descriptor to extract the data from it. To transmit another buffer, simply set the TX buffer descriptor R bit, RX buffer descriptor E bit, and STR bit in the SPCOM register.
3. Clear the SPI bit in the CISR.
4. Execute the **rfi** instruction.

16.13 THE I²C CONTROLLER

The inter-integrated circuits (I²C®) controller enables the MPC823 to exchange data with a number of other I²C devices, such as microcontrollers, EEPROMs, real-time clock devices, A/D converters, and LCD displays. The I²C controller is a synchronous, multimaster bus that is used to connect several integrated circuits on a board. It uses two wires—serial data (SDA) and serial clock (SCL)—to carry information between the integrated circuits that are connected to it.

The I²C controller consists of transmitter and receiver sections, an independent baud rate generator, and a control unit. The transmitter and receiver sections use the same clock, which is derived from the I²C baud rate generator in master mode and generated externally in slave mode. According to the I²C specification, wait states are inserted during a data transfer if the SCL signal is held low by a slave device. As a master in the middle of a data transfer, the I²C controller recognizes wait states by monitoring the SCL signal. The I²C controller does not begin counting down from a specific timeout value when SCL is asserted, so the software should monitor SCL assertion times for bus timeout.

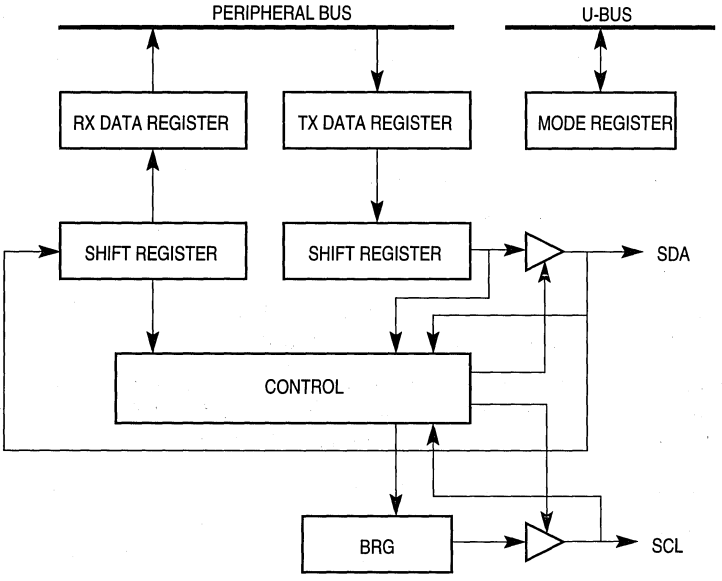


Figure 16-124. I²C Controller Block Diagram

The I²C receiver and transmitter are double-buffered, which corresponds to an effective FIFO size of 2 characters. The MPC823 I²C bit 0 (MSB) is shifted out first. When the I²C is not enabled in the I2MOD register, it consumes very little power.

16.13.1 Features

The following is a list of the I²C controller's main features:

- Two-Pin Interface
- Full-Duplex Operation
- Master or Slave I²C Mode Support
- MultiMaster Environment Support
- Continuous Transfer Mode for Autoscanning of a Peripheral
- Supports Maximum Capacitive Load of 400$_P$F on Both Bus Lines (Fully I²C Compliant)
- Independent Programmable Baud Rate Generator
- Supports I²C Low and High Speed Operation
- Supports 7-Bit I²C Addressing
- Open-Drain Output Pins Support MultiMaster Configuration
- Local Loopback Capability for Testing

16.13.2 I²C Controller Clocking and Pin Functions

Both the serial data (SDA) and serial clock (SCL) are bidirectional pins that need to be connected to a positive 5V power supply via an external pull-up resistor in the 6.8K Ohm to 10K Ohm range. Both pins are high when the I²C bus is free. The SCL signal clocks in received data and clocks out transmitted data on the SDA pin.

The I²C controller can be configured as a master or slave. When configured as a master, the I²C controller generates SCL, and then initiates and terminates the I/O operation. In addition, the I²C controller generates the SCL signal via a dedicated baud rate generator that takes its input from BRGCLK, which is described in **Section 5.3.4.2 The Baud Rate Generator Clock**. When configured as a slave, the I²C controller receives SCL as an input.

An I²C transaction is initiated when the master generates a start condition, which is defined as the SDA signal making a high-to-low transition while SCL is high. An acknowledge (ACK) is generated by the I²C receiver after each byte transfer. The receiver signals an ACK by driving the SDA signal low during the SCL clock pulse immediately following each data byte transmission. The data and ACK signals are always sampled on the rising edge of SCL. If the receiver does not issue an ACK after a data byte is transmitted, the I²C master generates a stop condition and transmission stops. A stop condition is when the SDA signal makes a low to high transition while the SCL signal remains high, as illustrated in Figure 16-125.

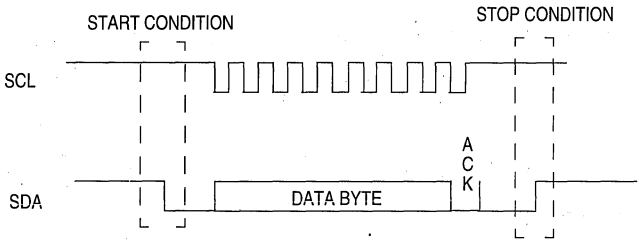


Figure 16-125. I²C Timing

16.13.3 I²C Controller Transmission and Reception Process

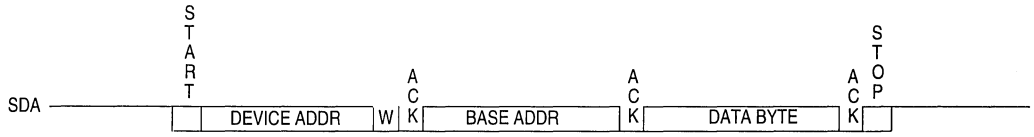
16.13.3.1 I²C MASTER MODE. The I²C controller, when functioning in master mode, initiates a transaction by transmitting a message specifying a read or write operation to the I²C slave. If a read operation is specified, the direction of the transfer is changed after the read operation is acknowledged, and the slave device then becomes the transmitter. If a write operation is specified, the direction of the transfer remains unchanged.

As the I²C controller shifts out each bit, it monitors the level of the SDA pin to detect a possible collision with other I²C master transmitters. If a collision is detected, transmission stops and the controller reverts to slave mode. A maskable interrupt may be issued to the core to allow the software to retransmit later.

In master mode operation, setting the S bit in the TX buffer descriptor control and status field will cause a start condition to be sent before this buffer is transmitted. If the TX buffer descriptor is the first one in the ring, then a start condition will be issued regardless of the S bit setting. Setting the L bit will cause a stop condition to be sent after the buffer is transmitted. You must set the L bit for the last TX buffer in the ring.

You must the set M/S bit in the I2COM register to configure the controller as a master. Clear the I2CMOD register's EN bit to disable the I²C controller before programming the SCL clock frequency you want with the I2MOD and I2BRG registers. The I2ADD register does not need to be programmed when you are operating the I²C controller in single-master mode. Enable the I²C controller by setting the EN bit in the I2MOD register.

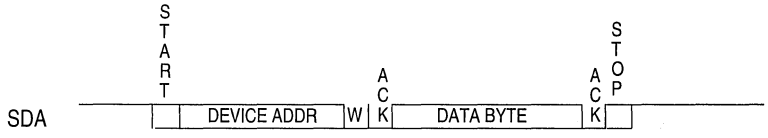
16.13.3.1.1 Master Write. To begin a write operation to a slave device that contains internal addresses, you need to prepare a TX data buffer that is N+2 bytes in length. The first byte contains the 7-bit device address, followed by the write bit asserted (R/$\overline{W}$ = 0). The second byte contains the internal base address of the write. The remaining N bytes contain the data to be written to the slave. The slave device will process the N bytes of data sequentially, starting at the specified base address.



NOTE: DATA AND ACK ARE REPEATED N TIMES.

Figure 16-126. Byte Write to Device with Internal Addresses

To begin a master write operation to a slave device that does not contain internal addresses, you need to prepare a TX data buffer that is N+1 bytes in length. The first byte contains the 7-bit slave device address, followed by the write bit asserted (R/$\overline{W}$ = 0). The remaining N bytes contain the data to be written to the slave.



NOTE: DATA AND ACK ARE REPEATED N TIMES.

Figure 16-127. Byte Write to Device without Internal Addresses

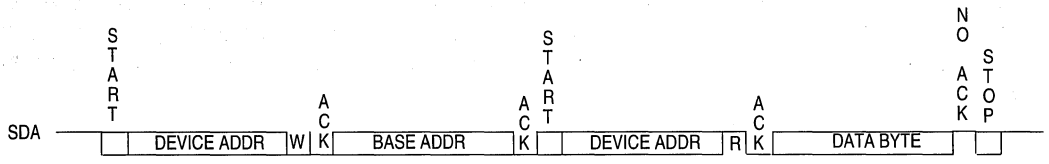
Next, when communicating to either slave device, the TX buffer descriptor's control and status field must have the W and L bits set. Set the I bit in the TX buffer descriptor to enable the transmission status to be updated in the I2CE register, and to enable I²C interrupts to the core. Set the R-bit in the TX buffer descriptor to prepare the buffer for transmission. The final step is to set the STR bit in the I2COM register to initiate transmission. The data starts transmitting once the SDMA channel loads the transmit FIFO with data and the I²C bus is not busy.

Note: Some slave devices such as serial E2PROMs may have a minimum write cycle time. For these devices, the I²C controller must wait a specified minimum time after a write before initiating a subsequent read or write. The needed delay must be implemented in software.

16.13.3.1.2 Master Read. To begin a read operation to a slave device that contains internal addresses, you need to prepare two TX buffer descriptors. TX buffer descriptor 0 should be 2 bytes in length. TX buffer descriptor 1 should be N+1 bytes in length, where N is the number of bytes to be read sequentially from the slave device. You also need to prepare one or more receive buffers for the N bytes of data to receive from the slave device.

The first byte of TX buffer descriptor 0 should contain the 7-bit slave address followed by the write bit asserted (R/$\overline{W}$ = 0). The second byte of the first transmit buffer should contain the 1-byte internal address on the slave device. This is the base address on the slave device of the data to be read. The transmission of TX buffer descriptor 0 is commonly referred to as a dummy write. Its purpose is to select the slave device.

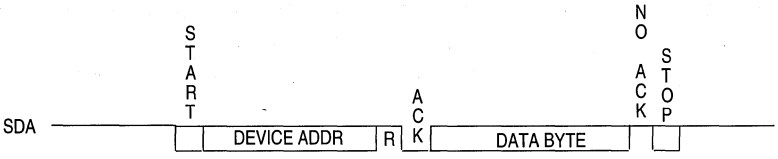
The first byte of the second TX buffer descriptor (1) should contain the slave address followed by the read bit asserted (R/$\overline{W}$ = 1). The remaining N bytes of TX buffer descriptor 1 may be uninitialized and merely serve as a place holder for the I²C controller. The TX buffer descriptor 1 control and status field should have the W, L, and S bits set.



NOTE: DATA AND ACK ARE REPEATED N TIMES.

Figure 16-128. Byte Read from Device with Internal Addresses

To begin a read operation to a slave device that does not contain internal addresses, you need to prepare TX buffer descriptor N+1 bytes in length, where N is the number of bytes to be read sequentially from the slave device. The first byte of the TX buffer descriptor should contain the 7-bit slave address followed by the read bit asserted (R/$\overline{W}$ = 1). You also need to prepare one or more receive buffers for the N bytes of data to receive from the slave device.



NOTE: DATA AND ACK ARE REPEATED N TIMES.

Figure 16-129. Byte Read from Device without Internal Addresses

When communicating to either slave device, set the R bit in the TX buffer descriptor(s) to prepare them for transmission. Set the I bit in the TX and RX buffer descriptors to enable the transmission and reception status to be updated in the I2CE register, and to enable I²C interrupts to the core. Set the E bit on the RX buffer descriptor(s) to prepare them for reception. Finally, set the STR bit in the I2COM register to initiate transmission. The data starts transmitting once the SDMA channel loads the transmit FIFO with data and the I²C bus is not busy.

16.13.3.1.3 I²C Loopback Configuration. Loopback on the I²C controller is a special case of master mode operation with a device that does not contain internal addresses. Refer to Figure 16-127 and Figure 16-129 for more information. To begin a loopback transmission, you need to prepare a TX buffer descriptor with a data buffer N+1 bytes in length, where N is the number of data bytes to be written back to the I²C controller. You also need to prepare one or more RX buffer descriptors to receive back the N bytes of data.

The first byte of the TX buffer descriptor should contain the address of the MPC823 I²C device's own address, as contained in the I2CADD register, followed by the write bit asserted (R/$\overline{W}$ = 0). The remaining N bytes of the TX buffer descriptor contain the data to be sent and received by the I²C controller.

Next, set the R bits on the TX buffer descriptor and the E bits in the RX buffer descriptors. The TX buffer descriptor control and status field should have the W and L bits set. The setting of the L bit will cause a stop condition to be issued after this buffer is transmitted to conclude the operation. Set the I bit in the TX and RX buffer descriptors to enable the transmission and reception status to be updated in the I2CE register, and to enable I²C transmit and receive interrupts to the core. You should then set the STR bit in the I2COM register to initiate the loopback operation.

16.13.3.2 I²C SLAVE MODE. When the I²C controller functions in slave mode, it receives messages from an I²C master and, in turn, sends back a reply. Once the I²C controller is configured for slave mode operation by clearing the M/S bit in the I2COM register, the SCL signal becomes an input driven by the external master. The I²C controller can operate with an SCL of any frequency from DC to beyond 400kHz.

After the start condition, the first transmitted byte to the I²C slave device contains the 7-bit slave device address and the read/write bit. The I²C controller will compare the transmitted slave device address with its own programmed address. If there is a match, the read/write bit is evaluated.

You must the clear the M/S bit in the I2COM register to configure the controller as a slave. You do not program the I2MOD and I2BRG registers to set the SCL frequency, as SCL is an input to the slave. Program the I2ADD register with the 7-bit I²C address of the slave. Enable the I²C controller by setting the EN bit in the I2MOD register.

16.13.3.2.1 Write to Master. If a write operation is requested by an external master, the I²C controller acknowledges the received data and writes it into a receive buffer using the next available RX buffer descriptor until the next start or stop condition is detected.

After transmitting each data byte, the master checks for the acknowledge bit from the slave. If an overrun condition occurs on the slave receiver, it will fail to acknowledge a byte and the transmission will abort. An overrun condition occurs when more data is transmitted than the slave can receive. A maskable interrupt may be issued by the I²C controller at the conclusion of a normal or errant reception.

To prepare the I²C controller in slave mode for data reception, you must configure one or more RX buffer descriptors to receive the data from the master. Set the E bit in the RX buffer descriptors to prepare them to receive data. Other RX buffer descriptor control bits, such as W and I, may be set as appropriate. You should then set the STR bit in the I2COM register to prepare the slave to respond to the master.

16.13.3.2.2 Read from Master. If a read operation is requested by an external master, the I²C controller will acknowledge the newly received byte containing the slave address and read bit (R/$\overline{W}$ = 1), only if the transmitter FIFO has been loaded by the SDMA channel.

If the transmitter is ready, the slave starts transmitting on the next clock pulse following the acknowledge. Otherwise, the transaction is aborted and a maskable TX error interrupt may be issued to notify the software to prepare data for transmission on another attempt. A maskable interrupt may be issued upon normal completion of transmission or after a other transmission errors occur. If an underrun condition occurs, the slave transmits ones until a stop condition is detected. An underrun occurs if the slave device does not transmit all of the data requested by the master.

To prepare the I²C controller in slave mode for data transmission, you must configure one or more TX buffer descriptors to transmit the data to the master. Set the E bit in the RX buffer descriptors to prepare them to receive data. Other TX buffer descriptor control bits, such as W, I, and L, may be set as appropriate. You should then set the STR bit in the I2COM register to prepare the slave to respond to the master.

16.13.4 I²C Parameter RAM Memory Map

The I²C controller parameter RAM area begins at the I²C base address, which is used for the general I²C parameters. It is similar to the SCC2 general-purpose parameter RAM. You must initialize certain parameter RAM values before the serial peripheral interface is enabled. The communication processor module initializes the other values. Once initialized, the parameter RAM values do not usually need to be accessed by your software. They should only be modified when there is no serial peripheral interface activity in progress.

Table 16-40. I²C Controller Parameter RAM Memory Map

ADDRESS	NAME	WIDTH	DESCRIPTION
I²C Base + 00	**RBASE**	Half-word	RX BD Base Address
I²C Base+ 02	**TBASE**	Half-word	TX BD Base Address
I²C Base+ 04	**RFCR**	Byte	RX Function Code
I²C Base+ 05	**TFCR**	Byte	TX Function Code
I²C Base+ 06	**MRBLR**	Half-word	Maximum Receive Buffer Length
I²C Base+ 08	RSTATE	Word	RX Internal State
I²C Base+ 0C	RPTR	Word	RX Internal Data Pointer
I²C Base+ 10	RBPTR	Half-word	RX BD Pointer
I²C Base+ 12	RCNT	Half-word	RX Internal Byte Count
I²C Base+ 14	RTMP	Word	RX Temp
I²C Base+ 18	TSTATE	Word	TX Internal State
I²C Base+ 1C	TPTR	Word	TX Internal Data Pointer
I²C Base+ 20	TBPTR	Half-word	TX BD Pointer
I²C Base+ 22	TCNT	Half-word	TX Internal Byte Count
I²C Base+ 24	TTMP	Word	TX Temp

NOTE: You are only responsible for initializing the items in bold. I²C Base = (IMMR & 0xFFFF0000) + 0x3C80.

- RBASE and TBASE—The RX and TX buffer descriptor base address entries are where the dual-port RAM starts receiving and transmitting data for the I²C buffer descriptors. They provide a great deal of flexibility in how buffer descriptors for an I²C controller are partitioned. By selecting RBASE and TBASE entries for the I²C controller and by setting the W bit in the last buffer descriptor in each buffer descriptor list, you can select how many buffer descriptors to allocate for the transmit and receive side of the I²C controller. However, you must initialize these entries before enabling the corresponding channel. Furthermore, you should not configure the buffer descriptor tables of the I²C controller to overlap because erratic operation will occur. RBASE and TBASE should contain a value that is divisible by eight.

- RFCR and TFCR—The RX and TX function code entries contain the value that you want to appear on the AT pins when the associated SDMA channel accesses memory. This register controls the byte-ordering convention used in the transfers.

RFCR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED			BO		AT		
RESET	0			0		0		
R/W	R/W			R/W		R/W		
ADDR	I2C BASE + 0x04							

Bits 0–2—Reserved

These bits are reserved and should be set to 0.

BO—Byte Ordering

You should set these bits to select the required byte ordering of the data buffer.

00 = The DEC/Intel convention is used for byte ordering (swapped operation) and is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation) is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT—Address Type 1–3

These bits contain the function code value used during the SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA-type access.

TFCR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED			BO		AT		
RESET	0			0		0		
R/W	R/W			R/W		R/W		
ADDR	I2C BASE + 0x05							

Bits 0–2—Reserved

These bits are reserved and should be set to 0.

BO—Byte Ordering

You should set this bit to select the required byte ordering of the data buffer.

00 = The DEC/Intel convention is used for byte ordering (swapped operation) and is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory.

01 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

1X = Motorola byte ordering (normal operation) is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.

AT—Address Type 1–3

These bits contain the function code value used during the SDMA channel memory access. AT0 is driven with a 1 to identify this SDMA channel access as a DMA-type access.

- MRBLR—The I^2C controller has a maximum receive buffer length register entry to define its receive buffer length and it defines the maximum number of bytes that the MPC823 writes to a receive buffer on that I^2C controller before moving to the next buffer. The MPC823 writes fewer bytes to the buffer than the MRBLR value if an error or end-of-frame occurs, but it never writes more bytes than the MRBLR value. Buffers you supply for the MPC823 to use should always be at least as long as MRBLR. The I^2C transmit buffers are not affected by the value you program into MRBLR and they can be individually chosen to have varying lengths, as needed. You choose the number of bytes to be transmitted by programming the DATA LENGTH field in the TX buffer descriptor.

  MRBLR is not intended to be dynamically changed while an I^2C controller is operating. However, if it is modified in a single bus cycle with one 16-bit move (not two 8-bit bus cycles back-to-back), a dynamic change in receive buffer length can be successfully achieved. This occurs when the communication processor module moves control to the next RX buffer descriptor in the table. Thus, a change to MRBLR does not have an immediate effect. To guarantee the exact RX buffer descriptor on which the change occurs, you should only change MRBLR when the I^2C receiver is disabled. The MRBLR value should be greater than zero and should be even if the character length of the data is greater than eight bits.

- RBPTR—The receive buffer pointer entry for each I²C channel points to the next buffer descriptor that the receiver transfers data to when it is in an idle state or to the current buffer descriptor during frame processing. After a reset or when the end of the buffer descriptor table is reached, the communication processor module initializes this pointer to the value programmed in the RBASE entry. Although in most applications you should not write the RBPTR, it can be modified when the receiver is disabled or when you are sure no receive buffer is currently in use.
- TBPTR—The transmit buffer pointer entry for each I²C channel points to the next buffer descriptor that the transmitter transfers data from when it is in an idle state or to the current buffer descriptor during frame transmission. After a reset or when the end of buffer descriptor table is reached, the communication processor module initializes this pointer to the value programmed in the TBASE entry. Although in most applications you should not write TBPTR, it can be modified when the transmitter is disabled or when you are sure no transmit buffer is currently in use.
- Other General Parameters—For normal operation, you do not need to access these parameters. They are only listed here because they provide helpful information for experienced users and they can be used for debugging purposes. Additional parameters are listed in Table 16-40. RPTR and TPTR are updated by the SDMA channels to show the next address to be accessed. TCNT is a down-count value that is initialized with the TX buffer descriptor data length and decremented with every byte read by the SDMA channels. RCNT is a down-count value that is initialized with the MRBLR value and decremented with every byte the SDMA channels write. The RSTATE, TSTATE, RTMP, TTMP, and reserved areas can only be used by the RISC microcontroller.

16.13.5 I²C Commands

You can program the CPM command register (CPCR) with the following commands to transmit or receive data.

- **INIT TX PARAMETERS**—This command initializes all transmit parameters in this serial channel's parameter RAM to their reset state and should only be issued when the transmitter is disabled. The **INIT TX AND RX PARAMS** command can also be used to reset the transmit and receive parameters.
- **CLOSE RX BD**—This command is used to force the I²C controller to close the current RX buffer descriptor if it is being used and to use the next buffer descriptor for any subsequently received data. If the I²C controller is not in the process of receiving data, no action is taken by this command.
- **INIT RX PARAMETERS**—This command initializes all the receive parameters in this serial channel's parameter RAM to their reset state and should only be issued when the receiver is disabled. The **INIT TX AND RX PARAMS** command can also be used to reset the receive and transmit parameters.

16.13.6 The I²C Buffer Descriptor Ring

The data associated with the I²C controller is stored in buffers, which are referenced by buffer descriptors organized in a buffer descriptor ring located in the dual-port RAM. This ring has the same basic configuration as the serial communication and serial management controllers.

The buffer descriptor ring forms a circular queue that helps you arrange the buffers you want to transmit or receive. Using the buffer descriptors, the communication processor module confirms reception and transmission or indicates error conditions so that the processor knows the buffers have been serviced. The actual buffers can reside in either external memory or internal memory and the data buffers can reside in the parameter RAM area of another controller if it is not enabled.

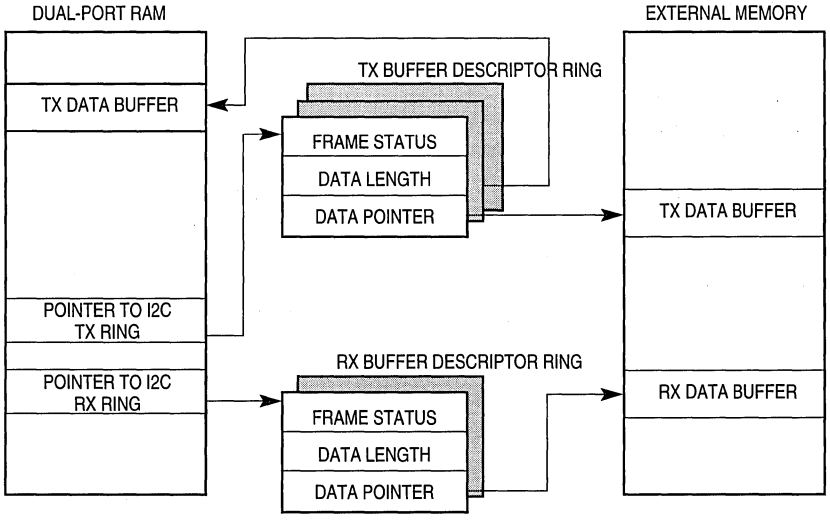


Figure 16-130. I²C Memory Format

16.13.7 Programming the I²C Controller

16.13.7.1 I²C MODE REGISTER. The read/write I²C mode (I2MOD) register controls both the I²C operation mode and clock source.

I2MOD

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED		REVD	GCD	FLT	PDIV		EN
RESET	0		0	0	0	0		0
R/W	R/W		R/W	R/W	R/W	R/W		R/W
ADDR	(IMMR & 0xFFFF0000) + 0x860							

Bits 0–1—Reserved

These bits are reserved and should be set to 0.

REVD—Reverse Data

This bit determines the receive and transmit character bit order.

0= Normal operation. Most-significant bit of character transmitted and received first.
1= Reverse data. Least-significant bit of character transmitted and received first.

GCD—General Call Disable

This bit determines if the receiver will acknowledge a general call address.

0= General call address is enabled.
1= General call address is disabled.

FLT—Clock Filter

This bit determines if the I²C input clock is filtered to prevent spikes in a noisy environment.

0 = I2CLK is not filtered.
1 = I2CLK is filtered by a digital filter.

PDIV—Pre Divider

This field determines the division factor of the clock before it is fed into the baud rate generator. The clock source for the I²C controller is the BRGCLK that is generated by the system interface unit.

00= Use the BRGCLK/32 as the input to the I²C baud rate generator.
01= Use the BRGCLK/16 as the input to the I²C baud rate generator.
10= Use the BRGCLK/8 as the input to the I²C baud rate generator.
11= Use the BRGCLK/4 as the input to the I²C baud rate generator.

EN—Enable I²C

This bit enables I²C operation. When EN is cleared, the I²C controller is in a reset state and consumes minimal power.

0 = I²C controller is disabled.
1 = I²C controller is enabled.

Note: Do not modify other bits of the I2MOD register when the EN bit is set or else erratic operation will occur.

16.13.7.2 I²C RECEIVE BUFFER DESCRIPTOR. Using receive (RX) buffer descriptors, the communication processor module reports information about each buffer of received data, closes the current buffer, generates a maskable interrupt, and starts receiving data in the next buffer when the current buffer is full. In addition, it closes the buffer when a stop or start condition is found on the I²C bus or when an overrun error occurs.

The first word of the RX buffer descriptor contains status and control bits, which you prepare before reception and then the communication processor module sets them after the buffer is closed. The second word contains the data length (in bytes) that is received and the third and fourth words contain a pointer that always points to the beginning of the received data buffer. The core should write these RX buffer descriptor bits before the I²C controller is enabled.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	**E**	RES	**W**	**I**	L	RESERVED									OV	RES
OFFSET + 2	DATA LENGTH															
OFFSET + 4	**RX DATA BUFFER POINTER**															
OFFSET + 6																

NOTE: You are only responsible for initializing the items in bold.

E—Empty

0 = The data buffer associated with this RX buffer descriptor is filled with received data or data reception is aborted due to an error condition. The core is free to examine or write to any fields of this RX buffer descriptor. The communication processor module does not use this buffer descriptor as long as the E bit is zero.

1 = The data buffer associated with this buffer descriptor is empty or reception is currently in progress. This RX buffer descriptor and its associated receive buffer are owned by the communication processor module. Once the E bit is set, the core should not write any fields of this RX buffer descriptor.

Bits 1, 5–13, and 15—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0 = This is not the last buffer descriptor in the RX buffer descriptor table.

1 = This is the last buffer descriptor in the RX buffer descriptor table. After this buffer is used, the communication processor module receives incoming data into the first buffer descriptor that RBASE points to in the table. The number of RX buffer descriptors in this table is programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0 = No interrupt is generated after this buffer is filled.

1 = The RXB bit in the I2CE register is set when this buffer is completely filled by the communication processor module, indicating the need for the core to process the buffer. The RXB bit can cause an interrupt if it is enabled.

L—Last

The I^2C controller sets this bit when the buffer is closed because a stop (or start) condition occurred on the bus or as a result of an overrun. The I^2C controller writes this bit after the received data is placed into the associated data buffer.

0 = This buffer does not contain the last character of the message.

1 = This buffer contains the last character of the message.

OV—Overrun

This bit indicates that a receiver overrun has occurred during reception. The I^2C controller writes this bit after the received data is placed into the associated data buffer.

DATA LENGTH

This field represents the number of octets that the communication processor module writes into this buffer descriptor data buffer. The communication processor module writes it once the buffer descriptor closes. The I²C controller writes these bits after the received data is placed into the associated data buffer. The actual amount of memory allocated for this buffer should be greater than or equal to the MRBLR.

RX DATA BUFFER POINTER

This field always points to the first location of the associated data buffer, must be even, and can reside in internal or external memory. The I²C controller writes these bits after the received data is placed into the associated data buffer.

16.13.7.3 I²C TRANSMIT BUFFER DESCRIPTOR. Data to be transmitted with the I²C is sent to the communication processor module by arranging it in buffers referenced by the transmit (TX) buffer descriptor ring. The first word of the TX buffer descriptor contains status and control bits. You should prepare the following bits before transmitting data.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	**R**	RES	**W**	**I**	**L**	**S**	RESERVED							NAK	UN	CL
OFFSET + 2	DATA LENGTH															
OFFSET + 4	**TX DATA BUFFER POINTER**															
OFFSET + 6																

NOTE: You are only responsible for initializing the items in bold.

Note: The communication processor module sets all the status bits in this buffer descriptor. You should clear all the status bits before submitting the buffer descriptor to the communication processor module. For example, the parity error bit is only set when a parity error occurs.

R—Ready

0 = The data buffer associated with this buffer descriptor is not ready for transmission. You are free to manipulate this buffer descriptor or its associated data buffer. The communication processor module clears this bit after the buffer is transmitted or after an error occurs.

1 = The data buffer, which you prepare for transmission, is not transmitted yet or is currently being transmitted. You cannot write any fields of this buffer descriptor once this bit is set.

Bits 1 and 6–12—Reserved

These bits are reserved and should be set to 0.

W—Wrap (Final Buffer Descriptor in Table)

0= This is not the last buffer descriptor in the TX buffer descriptor table.
1= This is the last buffer descriptor in the TX buffer descriptor table. After this buffer is used, the communication processor module receives incoming data into the first buffer descriptor that TBASE points to in the table. The number of TX buffer descriptors in this table is programmable and determined only by the W bit and overall space constraints of the dual-port RAM.

I—Interrupt

0= No interrupt is generated after this buffer is serviced.
1= The TXB or TXE bit in the event register is set when this buffer is serviced. TXB and TXE can cause interrupts if they are enabled.

L—Last

0= This buffer does not contain the last character of the message.
1= This buffer contains the last character of the message.

S—Transmit Start Condition

When this bit is set to 1, the I^2C controller transmits a start condition before the first byte of the buffer. If this buffer descriptor is the first one in the frame, a start condition is transmitted regardless of the value of this bit. This bit provides the ability to transmit a start byte or back-to-back frames.

0= A start condition is not transmitted before the first byte of the buffer, unless it is the first byte of a frame.
1= A start condition is transmitted before the first byte of the buffer.

NAK—No Acknowledge

This bit indicates that the transmission has been aborted because the last transmitted byte was not acknowledged. The I^2C controller writes this bit after it finishes transmitting the associated data buffer.

UN—Underrun

This bit indicates that the I^2C controller has encountered a transmitter underrun condition while transmitting the associated data buffer. The I^2C controller writes this bit after it finishes transmitting the associated data buffer.

CL—Collision

This bit indicates that transmission has been aborted because the transmitter was lost while arbitrating for the bus. The I^2C controller writes this bit after it finishes transmitting the associated data buffer.

DATA LENGTH

This field represents the number of octets the communication processor module should transmit from this buffer descriptor data buffer. However, it is never modified by the communication processor module. Normally, this value should be greater than zero. The I²C controller writes these bits after it finishes transmitting the associated data buffer.

TX DATA BUFFER POINTER

This field always points to the first location of the associated data buffer. They can be even or odd, unless the number of actual data bits in the character is greater than 8 bits, in which case the transmit buffer pointer must be even. The buffer can reside in internal or external memory. The I²C controller writes these bits after it finishes transmitting the associated data buffer.

16.13.7.4 I²C ADDRESS REGISTER. The 8-bit, memory-mapped, read/write I²C address (I2ADD) register holds the address for this I²C port. You need to program this register if you are operating in multimaster, slave, or local loopback mode.

I2ADD

BIT	0	1	2	3	4	5	6	7
FIELD	SAD							RESERVED
RESET	0							0
R/W	R/W							R/W
ADDR	(IMMR & 0xFFFF0000) + 0xA64							

SAD—Slave Address 0–6

This field holds the slave address for the I²C port.

Bit 7—Reserved

This bit is reserved and should be set to 0.

16.13.7.5 I²C BAUD RATE GENERATOR REGISTER. The 8-bit, memory mapped, read/write I²C baud rate generator (I2BRG) register sets the divide ratio of the baud rate generator. This register is set to all ones at hard reset.

I2BRG

BIT	0	1	2	3	4	5	6	7
FIELD	DIV							
RESET	1	1	1	1	1	1	1	1
R/W	R/W							
ADDR	(IMMR & 0xFFFF0000) + 0x868							

DIV—Division Ratio 0–7

This field specifies the divide ratio of the baud rate generator divider in the I²C clock generator. The output of the prescaler is divided by 2 x (DIV + 3 + (2 x FLT)) and the clock has a 50% duty cycle. The FLT bit is in the I2MOD register.

Note: The minimum value for DIV is three if the digital filter is enabled and six of it is enabled.

16.13.7.6 I²C COMMAND REGISTER. The 8-bit read/write I²C command (I2COM) register is used to start I²C operation.

I2COM

BIT	0	1	2	3	4	5	6	7
FIELD	STR	RESERVED						M/S
RESET	0	0						0
R/W	R/W	R/W						R/W
ADDR	(IMMR & 0xFFFF0000) + 0x86C							

STR—Start Transmit

When the I²C controller is in master mode, setting this bit to 1 causes the I²C controller to start transmitting data from the I²C transmit buffers if they are ready. When the I²C controller is in slave mode, setting the STR bit to 1 when the I²C controller is idle causes it to load the transmit data register from the I²C transmit buffer and start transmitting when it receives an address byte that matches the slave address with the R/$\overline{W}$ bit set to 1. The STR bit is always read as a zero.

Bits 1–6—Reserved.
These bits are reserved and should be set to 0.

M/S—Master/Slave
This bit configures the I²C controller to operate as a master or a slave.

0 = I²C controller is a slave.
1 = I²C controller is a master.

16.13.7.7 I²C EVENT REGISTER. The 8-bit memory-mapped I²C event register (I2CER) is used to generate interrupts and report events recognized by the I²C controller. When an event is recognized, the I²C controller sets its corresponding bit in the I2CER. Interrupts generated by this register can be masked in the I²C mask register. A bit is cleared by writing a 1 (writing a zero has no effect) and more than one bit can be cleared at a time. All unmasked bits must be cleared before the communication processor module clears the internal interrupt request. This register is cleared by reset and can be read at any time.

I2CER

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED			TXE	RESERVED	BSY	TXB	RXB
RESET	0			0	0	0	0	0
R/W	R/W			R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x870							

Bits 0–2 and 4—Reserved
These bits are reserved and should be set to 0.

TXE—TX Error
This bit indicates that an error has occurred during transmission.

BSY—Busy Condition
This bit indicates that received data has been discarded due to a lack of buffers. This bit is set after the first character is received for which there is no receive buffer available.

TXB—TX Buffer
This bit indicates that a buffer has been transmitted. It is set once the transmit data of the last character in the buffer is written to the transmit FIFO. You must wait two character times to be sure that the data is completely sent over the transmit pin.

RXB—RX Buffer
This bit indicates that a buffer has been received. This bit is set after the last character is written to the receive buffer and the RX buffer descriptor is closed.

16.13.7.8 I²C MASK REGISTER. The 8-bit read/write I²C mask register (I2CMR) has the same bit formats as the I2CER. If a bit in the I2CMR is 1, the corresponding interrupt in the I2CER is enabled. If the bit is zero, the corresponding interrupt in the I2CER is masked. This register is cleared by reset.

I2CMR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED			TXE	RESERVED	BSY	TXB	RXB
RESET	0			0	0	0	0	0
R/W	R/W			R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x874							

16.13.8 I²C Controller Initialization Sequence

The following initialization sequence is for the I²C controller to operate in master mode, and read one byte from a slave device that contains an internal read address. The I²C controller operates the SCL at 391kHz. A system frequency of 50MHz is assumed. The SDA and SCL pins of the MPC823 are connected to an external 5V power supply with 6.8K Ohm to 10K Ohm resistors.

1. Configure the port B pins to enable the SDA and SCL pins. Write PBPAR, PBDIR, and PBODR bits 26 and 27 with ones.
2. Disable the I²C controller by clearing the I2MOD register, including the EN bit. Now you can modify other fields in the I2MOD register.
3. Configure the I²C dedicated baud rate generator to operate at 391kHz at a system frequency of 50MHz by programming the divider and pre-divider. The overall I²C baud rate generator clock divider is 128 (decimal), and is realized by establishing a pre-divider of 8 and a divider of 16. Write the PDIV field of the I2MOD register with 2 to pre-divide by 8. Write the DIV field of the I2BRG register with 5 to divide by 16.
4. Write 0x01 to the I2COM register to configure controller for master mode operation.
5. Write 0x0001 to the SDCR to set the SDMA bus arbitration level to 5.
6. Write RBASE and TBASE in the I²C parameter RAM to point to the RX and TX buffer descriptors in the dual-port RAM. Assuming the initial RX buffer descriptor at the beginning of dual-port RAM and the initial TX buffer descriptor 64 bytes from the beginning, write RBASE with 0x2000 and TBASE with 0x2040.
7. Write 0x11 into the CPCR to execute the INIT RX AND TX PARAMS command for I²C.
8. Write 0x15 into the RFCR and TFCR for normal operation.
9. Write MRBLR with the maximum bytes per receive buffer. For this case, assume 16 bytes, so MRBLR = 0x10.
10. Initialize the RX buffer descriptor. Assume the RX data buffer is at 0x00001000 in main memory. Write 0xB000 to RX_BD_Status, 0x0 to RX_BD_Length (optional), and 0x00001000 to RX_BD_Pointer.

11. Initialize the TX buffer descriptors. Assume the TX data buffer 0 is at 0x00004000 in main memory. Write 0x9000 to TX_BD_Status 0, 0x2 to TX_BD_Length 0, and 0x00004000 to TX_BD_Pointer 0. Assume the TX data buffer 1 is at 0x00004010 in main memory. Write 0xBC000 to TX_BD_Status 1, 0x2 to TX_BD_Length 1, and 0x00004010 to TX_BD_Pointer 1.
12. Write 0xffff to the I2CER to clear any previous events.
13. Write 0x17 to the I2CMR to enable all transmit and receive interrupts.
14. Write 0x10000 to the CIMR to allow I^2C to generate a system interrupt. Write 0x8080 to the CICR to configure interrupt request level 4 and enable CPM interrupts.
15. Write 0x01 to the I2MOD register to enable the I^2C controller.
16. Set the STR bit in the I2COM register to begin the master read operation.

16.14 THE PARALLEL I/O PORTS

The communication processor module supports four general-purpose I/O ports—A, B, C, and D. Each pin in the I/O ports can be configured as a general-purpose I/O pin or as a dedicated peripheral interface pin. Port A is shared with the serial communication controller RXD2 and TXD2 pins, the bank of clocks pins, and some time-division multiplex pins. Port B is shared with functions like the IDMA, SMC, SPI, USB, and I^2C pins. Port C is shared with the $\overline{RTS}$, $\overline{CTS}$, and $\overline{CD}$ pins of the serial communication controllers as well as some time-division multiplex pins. However, port C is unique in that its pins can generate interrupts to the CPM interrupt controller.

You can configure the pins as an input or output, to have a latch for data output, or to be read or written at any time. You can even configure them to be either general-purpose I/O or dedicated peripheral pins. Ports A and B have pins that can be configured as open-drain. These pins drive a zero voltage, but they three-state when driving a high voltage.

Note: The port pins do not have internal pull-up resistors. Because of the significant flexibility of the MPC823 communication processor module, many dedicated peripheral functions are multiplexed onto ports A, B, and C. The functions are grouped in such a way as to maximize the usefulness of the pins in the greatest number of MPC823 applications. It may be difficult to fully understand the pin assignment capability described in this section until you better understand the CPM peripherals themselves.

16.14.1 Features

The following is a list of the parallel I/O port's main features:

- Port A is 12 Bits
- Port B is 16 Bits
- Port C is 12 Bits
- Port D is 13 Bits
- All Ports Are Bidirectional
- All Ports Have Alternate On-Chip Peripheral Functions
- All Ports Are Three-Stated at System Reset
- All Pin Values can Be Read While the Pin Is Connected to an On-Chip Peripheral
- Ports A and B have Open-Drain Capability
- Port C has 12 Interrupt Input Pins

16.14.2 Port A Pin Functionality

The 12 port A pins are independently configured as general-purpose I/O pins if the corresponding bit in the port A pin assignment register (PAPAR) is cleared. On the other hand, each pin is configured as a dedicated on-chip peripheral pin if the corresponding PAPAR bit is set. When the port A pin is configured as a general-purpose I/O pin, the signal direction for that pin is determined by the corresponding control bit in the port A data direction register (PADIR). The port A I/O pin is configured as an input if the corresponding PADIR bit is cleared and configured as an output if the corresponding bit is set. All PAPAR and PADIR bits are cleared at system reset, thus configuring all port A pins as general-purpose input pins.

Table 16-41 contains the default description of all port A pin options.

Table 16-41. Port A Pin Assignment

SIGNAL	PIN FUNCTION			
	PAPAR = 0	PAPAR = 1		PERIPHERALS
		PADIR = 0	PADIR = 1	
PA15	PORT A15	USBRXD	—	GND
PA14	PORT A14	**USBOE**	—	—
PA13	PORT A13	RXD2	—	GND
PA12	PORT A12	**TXD2**	—	—
PA9	PORT A9	SMRXD2	**L1TXDA**	Undefined
PA8	PORT A8	SMTXD2	L1RXDA	GND
PA7	PORT A7	CLK1/TIN1/L1RCLKA	BRGO1	CLK1/TIN1/L1RCLKA = BRGO1
PA6	PORT A6	CLK2	TOUT1	CLK2 = GND
PA5	PORT A5	CLK3/TIN2/L1TCLKA	BRGO2	CLK3/TIN2/L1TCLKA = BRGO2
PA4	PORT A4	CLK4	TOUT2	CLK4 = GND

NOTE: The items in bold have open-drain capability.
The I/O direction for the TDM and SMC2 pins are reversed. See PA8 and PA9.

If a port A pin is selected as a general-purpose I/O pin, it can be accessed through the port A data register (PADAT). Data written to the PADAT is stored in an output latch. If a port A pin is configured as an output, the output latch data is gated onto the port pin. When PADAT is read, the port pin itself is read. If a port A pin is configured as an input, data written to PADAT is still stored in the output latch, but is prevented from reaching the port pin. In this case, when PADAT is read, the state of the port pin is read. If an input to a peripheral is not supplied from a pin, then a default value is supplied to the on-chip peripheral as listed in Table 16-41.

16.14.3 The Port A Registers

Port A has four 16-bit, memory-mapped, read/write control registers.

16.14.3.1 PORT A OPEN-DRAIN REGISTER. The port A open-drain register (PAODR) indicates when the port pins are configured in a normal or wired-OR configuration. Three of the PAODR bits can be open-drain to correspond to those pins that have serial channel output capability. The other bits are always zero. PAODR is cleared by system reset.

PAODR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED									OD9	0	0	OD12	0	OD14	0
RESET	0									0	0	0	0	0	0	0
R/W									R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x954															

Bits 0–8—Reserved

These bits are reserved and should be set to 0.

OD0–OD15—Open-Drain Pins 0-15

0 = The I/O pin is actively driven as an output.
1 = The I/O pin is an open-drain driver. As an output, the pin is actively driven low. Otherwise, it is three-stated.

16.14.3.2 PORT A DATA REGISTER. A read of the port A data (PADAT) register returns the data at the pin, regardless of whether the pin is defined as an input or output. This allows output conflicts to be found on the pin by comparing the written data with the data on the pin. A write to the PADIR is latched and if the bit is configured as an output, the value latched for that bit is driven onto its respective pin. PADAT is not initialized and is undefined at reset.

PADAT

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED				D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
RESET	0				0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W				R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x956															

Bits 0–3—Reserved

These bits are reserved and should be set to 0.

D4–D15—Data Pins 4-15

The value written into this field may be read on the Port A pins.

16.14.3.3 PORT A DATA DIRECTION REGISTER. The port A data direction (PADIR) register is cleared by system reset.

PADIR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED				DR4	DR5	DR6	DR7	DR8	DR9	DR10	DR11	DR12	DR13	DR14	DR15
RESET	0				0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W				R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x950															

Bits 0–3—Reserved

These bits are reserved and should be set to 0.

DR4–DR15—Pin 4-15 Direction

0 = The corresponding pin is an input.
1 = The corresponding pin is an output.

16.14.3.4 PORT A PIN ASSIGNMENT REGISTER. The port A pin assignment register (PAPAR) is cleared at system reset.

PAPAR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED				DD4	DD5	DD6	DD7	DD8	DD9	DD10	DD11	DD12	DD13	DD14	DD15
RESET	0				0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W				R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x952															

Bits 0–3—Reserved

These bits are reserved and should be set to 0.

DD4–DD15—Dedicated Peripheral Pins 4-15

0 = General-purpose I/O. The peripheral functions of the pin are not used.
1 = Dedicated peripheral function. The pin is used by the internal module. The on-chip peripheral function to which it is dedicated can be determined by other bits.

16.14.4 Port A Example Configurations

The port A pins can be configured in many ways. PA15 can be configured as a general-purpose I/O pin, but not an open-drain pin. It can also be the USBRXD pin for the USB. If it is configured as a general-purpose I/O pin, the USBRXD input is internally grounded. Figure 16-131 and Figure 16-132 illustrate the block diagrams of the PA15 and PA14 pins.

PA14 can be configured as a general-purpose I/O pin, either open-drain or not. It can also be the USBTXD pin for the USB. If USBTXD is configured as an output on PA14 and the OD14 bit is set in the PAODR, then USBTXD is output from the USB as an open-drain output. If PA14 is configured as a general-purpose I/O pin, then the USBTXD output is not externally connected.

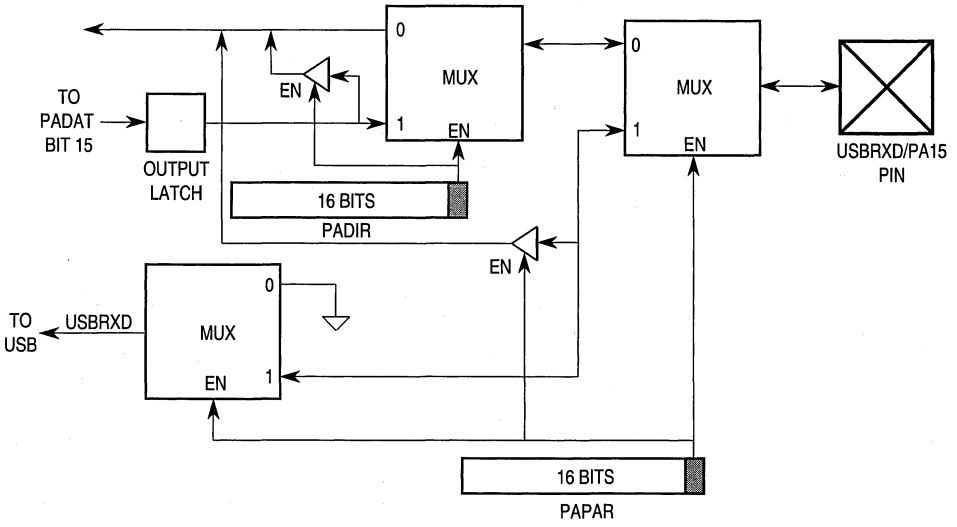


Figure 16-131. Parallel Block Diagram For PA15

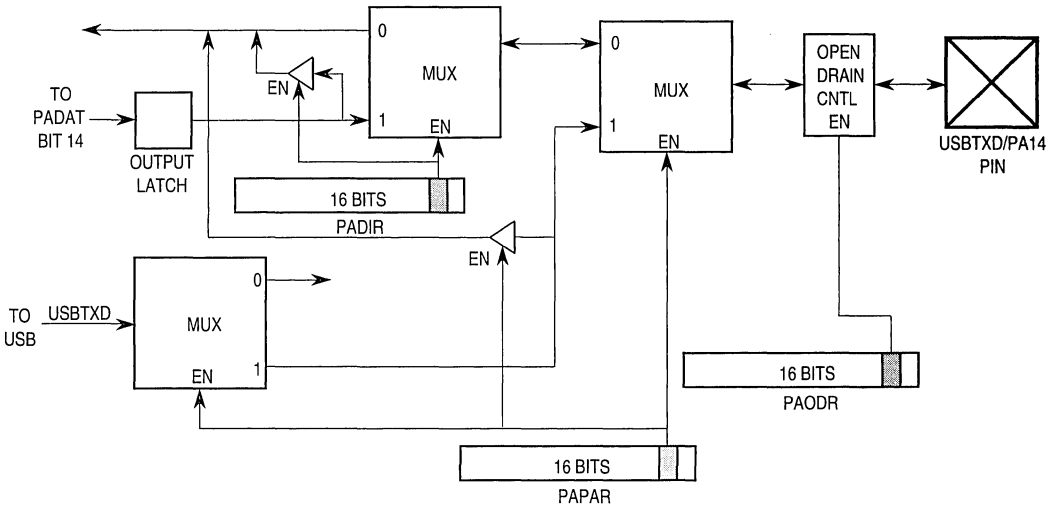


Figure 16-132. Parallel Block Diagram For PA14

PA7 can be configured as a general-purpose I/O pin, but not an open-drain pin. If the corresponding PADIR bit is a zero, it can also be the CLK1 pin, the TIN1 pin, the L1RCLKA pin, or all three at once. There is nothing to select other than these three inputs in port A because the connections are made separately in the serial interface and timer mode registers. If the PADIR bit is a 1, this pin can also be the BRGO1 pin. If the PA7 pin is a general-purpose I/O pin, then the input to the on-chip peripheral is internally connected to BRG01. Refer to **Section 16.7 The Serial Interface with Time-Slot Assigner** for more details about using the CLK1 and L1RCLKA pins.

PA4 can be configured as a general-purpose I/O pin, but not an open-drain pin. If the PADIR bit is zero, PA4 can also be the CLK4 pin. If the PADIR bit is a 1, PA4 can be the $\overline{\text{TOUT2}}$ pin. If the PA4 pin is a general-purpose I/O pin, then the input to the on-chip CLK4 function. This option is useful because not all CLK pins can be routed to all serial channels in every situation. The ability to send a clock from CLK8 to CLK4 increases the flexibility of this assignment process. Refer to **Section 16.7 The Serial Interface with Time-Slot Assigner** for more details.

16.14.5 Port B Pin Functionality

All port B pins can be open-drain and are independently configured as general-purpose I/O pins if the corresponding bit in the PBPAR is cleared. They are configured as dedicated on-chip peripheral pins if the corresponding PBPAR bit is set. When configured as a general-purpose I/O pin, the signal direction of that pin is determined by the corresponding control bit in the PBDIR. The port I/O pin is configured as an input if the corresponding PBDIR bit is cleared and as an output if the corresponding PBDIR bit is set. All PBPAR bits and PBDIR bits are cleared by total system reset, thus configuring all port B pins as general-purpose input pins. Refer to Table 16-42 for a description of all port B pin options.

If a port B pin is used as a general-purpose I/O pin, it can be accessed with the PBDAT where data is stored in an output latch. If a port B pin is configured as an output, the output latch data is gated onto the port pin. When PBDAT is read, the port pin itself is read. If a port B pin is configured as an input, data written to PBDAT is still stored in the output latch, but is prevented from reaching the port pin. When PBDAT is read, the state of the port pin is read. Many of the port B pins have more than one function, including on-chip peripheral functions for SPI, I^2C, SMC1, SMC2, TDMa, and LCD. PB27 and PB26 are unusual in that their on-chip peripheral functions (BRGO2 and BRGO1) are also used in port A. This allows an alternate way to output the BRGO pins if other functions are used. PB18 and PB16 are also unusual in that their on-chip peripheral functions (RTS2 and L1RQA) are used in port C, which is an alternate location to output these pins if using other functions on port C.

Table 16-42. Port B Pin Assignment

SIGNAL	PIN FUNCTION			
	PBPAR = 0	PBPAR = 1		INPUT TO ON-CHIP PERIPHERALS
		PBDIR = 0	PBDIR = 1	
PB31	PORT B31	SPISEL	LCD_A	SPISEL=V_{DD}
PB30	PORT B30	—	SPICLK	SPICLK = GND
PB29	PORT B29	—	SPIMOSI	SPIMOSI=V_{DD}
PB28	PORT B28	—	SPIMISO	SPIMISO = SPIMOSI
PB27	PORT B27	BRGO1	I2CSDA	I2CSDA= V_{DD}
PB26	PORT B26	BRGO2	I2CSCL	I2CSCL = GND
PB25	PORT B25	SMTXD1	—	—
PB24	PORT B24	SMRXD1	—	—
PB23	PORT B23	SMSYN1	SDACK1	$\overline{\text{SMSYN1}}$ = GND
PB22	PORT B22	—	SDACK2	
PB19	PORT B19	L1ST1	LCD_B	—
PB18	PORT B18	L1ST2	RTS2	—
PB17	PORT B17	L1ST3	LCD_C	—
PB16	PORT B16	L1ST4	L1RQA	—

16.14.6 The Port B Registers

Port B has four 16-bit, memory-mapped, read-write control registers.

16.14.6.1 PORT B OPEN-DRAIN REGISTER. The 16-bit port B open drain register (PBODR) indicates when the port pins are configured in a a normal or wired-OR configuration. Bits 0 and 15 of this register are reserved.

PBODR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0xAC0															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	OD16	OD17	OD18	OD19	OD20	OD21	OD22	OD23	OD24	OD25	OD26	OD27	OD28	OD29	OD30	OD31
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xAC2															

OD16–OD31—Open-Drain Pins 16-31

0 = The I/O pin is actively driven as an output.
1 = The I/O pin is an open-drain driver. As an output, the pin is actively driven low. Otherwise, it is three-stated.

Note: SMTXD1 cannot be set as an open-drain driver, regardless of how this register is set.

16.14.6.2 PORT B DATA REGISTER. A read of the port B data register (PBDAT) returns the data to the pin, regardless of whether the pin is defined as an input or output. This allows output conflicts to be found on the pin by comparing the written data with the data on the pin. A write to the PBDAT is latched and if that bit in the PBDIR is configured as an output, the value latched for that bit is driven onto its respective pin. PBDAT can be read or written at any time, is not initialized, and is undefined at reset.

PBDAT

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0xAC4															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	D16	D17	D18	D19	D20	D21	D22	D23	D24	D25	D26	D27	D28	D29	D30	D31
RESET	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xAC6															

D16–31—Data Pins 16–31

These bits contain data can be read or written from the port B pins.

16.14.6.3 PORT B DATA DIRECTION REGISTER. The port B data direction register (PBDIR) is cleared at system reset.

PBDIR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0xAB8															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	DR16	DR17	DR18	DR19	DR20	DR21	DR22	DR23	DR24	DR25	DR26	DR27	DR28	DR29	DR30	DR31
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xABA															

DR14–DR31—Data Direction

0 = The corresponding pin is an input.
1 = The corresponding pin is an output.

16.14.6.4 PORT B PIN ASSIGNMENT REGISTER. The port B pin assignment register (PBPAR) is cleared by system reset.

PBPAR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
RESET	0															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0xABC															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	DD16	DD17	DD18	DD19	DD20	DD21	DD22	DD23	DD24	DD25	DD26	DD27	DD28	DD29	DD30	DD31
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xABE															

DD14–DD31—Dedicated Function

0 = General-purpose I/O. The peripheral functions of the pin are not used.
1 = Dedicated peripheral function. The pin is used by the internal module. The on-chip peripheral function to which it is dedicated can be determined by other bits such as those in the PBDIR.

16.14.7 Port B Configuration Example

You can configure the PB31 pin as a general-purpose I/O or open-drain pin. It can also be the LCD_A pin for the LCD controller or the SPI select input $\overline{\text{SPISEL}}$ pin. If PB31 is not configured to connect to the LCD_A or $\overline{\text{SPISEL}}$ signal, then the serial peripheral interface receives V_{DD} on that signal.

16.14.8 Port C Pin Functionality

Port C consists of 12 general-purpose I/O pins that have interrupt capability. Refer to Table 16-43 for a description of all port C pin options.

Table 16-43. Port C Pin Assignment

SIGNAL	PCPAR = 0		PCPAR = 1		INPUT TO ON-CHIP PERIPHERALS
	PCDIR = 1 OR PCSO = 0	PCDIR = 0 AND PCSO = 1	PCDIR = 0	PCDIR = 1	
PC15	Port C15	$\overline{\text{DREQ1}}$	—	L1ST5	EXT0 = VDD
PC14	Port C14	$\overline{\text{DREQ2}}$	$\overline{\text{RTS2}}$	L1ST6	EXT1 = VDD
PC13	Port C13	—	—	L1ST7	—
PC12	Port C12	—	L1RQA	L1ST8	—
PC11	Port C11	USBRXP	—		GND
PC10	Port C10	USBRXN	TGATE1		GND
PC9	Port C9	$\overline{\text{CTS2}}$	—		GND
PC8	Port C8	$\overline{\text{CD2}}$	TGATE1		GND
PC7	Port C7	—	—	USBTXP	—
PC6	Port C6	—	USBTXN		—
PC5	Port C5	—	L1TSYNCA	SDACK1	—
PC4	Port C4	—	L1RSYNCA		GND

All PCDIR and PCPAR bits are cleared by a total system reset, thus configuring all port pins as general-purpose input pins. Notice that the global CPM interrupt mask register is also cleared when a total system reset occurs, so if any port C pin is left floating it does not cause a false interrupt.

If a port C pin is selected as a general-purpose I/O pin, it can be accessed through the PCDAT register where written data is stored in an output latch. If a port C pin is configured as an output, the output latch data is gated onto the port pin. When the PCDAT register is read, the port pin itself is read. If a port C pin is configured as an input, data written to PCDAT register is still stored in the output latch, but is prevented from reaching the port pin. In this case, when PCDAT register is read, the state of the port pin is read.

When the pin is configured as an output, port C interrupts cannot be generated. To configure a port C pin as a general-purpose output pin, follow these steps:

1. Write the corresponding PCPAR bit with a zero.
2. Write the corresponding PCDIR bit with a 1.
3. Write the corresponding PCSO bit with a zero (for clarity).
4. The corresponding PCINT bit is a "don't care" bit.
5. Write the pin value using the PCDAT register.

To configure a port C pin as a general-purpose input pin that does not generate an interrupt, follow these steps:

1. Write the corresponding PCPAR bit with a zero.
2. Write the corresponding PCDIR bit with a zero.
3. Write the corresponding PCSO bit with a zero.
4. The corresponding PCINT bit is a "don't care" bit.
5. Write the corresponding CIMR bit with a zero to prevent interrupts from being generated to the core.
6. Read the pin value using the PCDAT register.

When a port C pin is configured as a general-purpose I/O input, a change in the port C interrupt register (PCINT) causes an interrupt request signal to be sent to the CPM interrupt controller. You can program each port C line to assert an interrupt request when a high-to-low change occurs or when any change occurs. Each port C line asserts a unique interrupt request to the CPM interrupt pending register and has a different internal interrupt priority level within the CPM interrupt controller. Refer to **Section 16.15 The CPM Interrupt Controller** for more details. Each request can be masked independently in the CPM interrupt mask register. To configure a port C pin as a general-purpose input pin that generates an interrupt, follow these steps:

1. Write the corresponding PCPAR bit with a zero.
2. Write the corresponding PCDIR bit with a zero.
3. Write the corresponding PCSO bit with a zero.
4. Set the PCINT bit to find out which edges cause the interrupts.
5. Write the corresponding CIMR bit with a 1 so that interrupts can be sent to the core.
6. Read the pin value using the PCDAT register.

The port C lines associated with the $\overline{CD2}$ and $\overline{CTS2}$ pins have a mode of operation in which the pin can be internally connected to the serial communication controller but can also generate interrupts. Port C still detects changes on the $\overline{CTS2}$ and $\overline{CD2}$ pins and asserts the corresponding interrupt request, but the serial communication controller simultaneously uses the $\overline{CTS2}$ and/or $\overline{CD2}$ pin to automatically control operation. This allows you to fully implement protocols V.24, X.21, and X.21 bis with the assistance of other general-purpose I/O lines. To configure a port C pin as a $\overline{CTS2}$ or $\overline{CD2}$ pin that connects to the serial communication controller and generates interrupts, follow these steps:

1. Write the corresponding PCPAR bit with a zero.
2. Write the corresponding PCDIR bit with a zero.
3. Write the corresponding PCSO bit with a 1.
4. Set the PCINT bit to find out which edges cause the interrupts.
5. Write the corresponding CIMR bit with a 1 so that interrupts can be sent to the core.
6. The pin value can be read at any time using the PCDAT register.

Note: After connecting the $\overline{CTS2}$ or $\overline{CD2}$ pins to the serial communication controller, you must also choose "normal operation" mode in the DIAG field of the GSMR_L to enable or disable SCC2 transmission and reception with these pins.

The $\overline{DREQ1}$ and $\overline{DREQ2}$ pins in port C can assert an external request to the RISC microcontroller instead of asserting an interrupt to the core. You can program each pin to assert an interrupt request when a high-to-low change occurs or any change that as configured in PCINT.

Note: Do not program the $\overline{DREQ1}$ and $\overline{DREQ2}$ pins to assert external requests to the RISC microcontroller, unless instructed to do so by Motorola as part of a RAM microcode package. Otherwise, erratic behavior will occur.

16.14.9 Port C Registers

You can communicate with the port C using five registers. The port C interrupt control register (PCINT) indicates how changes on the pin cause interrupts when they are generated with that pin. The port C special options register (PCSO) indicates whether certain port C pins can connect to on-chip peripherals and generate an interrupt at the same time. The remaining port C registers exist on the other ports as well. However, since port C does not have an open-drain capability, there is no open-drain register.

16.14.9.1 PORT C DATA REGISTER. A read of the port C data register (PCDAT) returns the data to the pin, regardless of whether the pin is defined as an input or output. This allows output conflicts to be found on the pin by comparing the written data with the data on the pin. A write to the PCDAT is latched and if that bit in the PCDIR is configured as an output, the value latched for that bit is driven onto its respective pin. PCDAT can be read or written at any time, is not initialized, and is undefined at reset.

PCDAT

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED				D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
RESET	0	0	0	0	—	—	—	—	—	—	—	—	—	—	—	—
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x966															

NOTE: — = Undefined.

Bits 0–3—Reserved

These bits are reserved and should be set to 0.

D4–15—Data Pins 4–15

These bits contain data can be read or written from the port C pins.

16.14.9.2 PORT C DATA DIRECTION REGISTER. The port C data direction register (PCDIR) is a 16-bit register that is cleared by system reset.

PCDIR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED				DR4	DR5	DR6	DR7	DR8	DR9	DR10	DR11	DR12	DR13	DR14	DR15
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x960															

Bits 0–3—Reserved

These bits are reserved and should be set to 0.

DR4–DR14—Data Direction Pins 4–14

0 = The corresponding pin is an input.
1 = The corresponding pin is an output.

16.14.9.3 PORT C PIN ASSIGNMENT REGISTER. The port C pin assignment register (PCPAR) is a 16-bit register that is cleared by system reset.

PCPAR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED				DD4	DD5	DD6	DD7	DD8	DD9	DD10	DD11	DD12	DD13	DD14	DD15
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x962															

Bits 0–3—Reserved

These bits are reserved and should be set to 0.

DD14–DD31—Dedicated Function

0 = General-purpose I/O. The peripheral functions of the pin are not used.
1 = Dedicated peripheral function. The pin is used by the internal module. The on-chip peripheral function to which it is dedicated can be determined by other bits such as those in the PBDIR.

16.14.9.4 PORT C SPECIAL OPTIONS REGISTER. The port C special options (PCSO) register is a 16-bit read/write register. Each bit defined in the PCSO corresponds to a port C line (PC8–PC11 and PC14–PC15). The PCSO is cleared by reset.

PCSO

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED								CD2	CTS2	RESERVED				DREQ1	DREQ2
RESET	0								0	0	0				0	0
R/W	R/W								R/W	R/W	R/W				R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x964															

Bits 0–7 and 10–13—Reserved

These bits are reserved and should be set to 0.

CD2—Carrier Detect

0 = PC8 is a general-purpose interrupt I/O pin. The SCC2 internal $\overline{CD2}$ signal is always asserted. If PCDIR configures this pin as an input, the pin can generate an interrupt to the core, as controlled by the PCINT bits.
1 = PC8 is connected to the corresponding SCC2 signal input in addition to being a general-purpose interrupt pin.

CTS2—Clear To Send

0 = PC9 is a general-purpose interrupt I/O pin. The SCC2 internal $\overline{\text{CTS2}}$ signal is always asserted. If PCDIR configures this pin as an input, the pin can generate an interrupt to the core, as controlled by the PCINT bits.
1 = PC9 is connected to the corresponding SCC2 signal input in addition to being a general-purpose interrupt pin.

DREQx— DMA Request to the RISC Microcontroller

0 = PCx is a general-purpose interrupt I/O pin, with the direction controlled in PCDIR. If PCDIR configures this pin as an input, the pin can generate an interrupt to the core, as controlled by the PCINT bits.
1 = PCx becomes an external request to the RISC microcontroller instead of being a general-purpose interrupt pin. The corresponding PCINT bits control when a request is generated.

Note: $\overline{\text{DREQx}}$ should only be set if you are using IDMA.

16.14.9.5 PORT C INTERRUPT CONTROL REGISTER. The16-bit read/write port C interrupt control (PCINT) register whose bits correspond to a port C line to determine whether that line asserts an interrupt request when a high-to-low change occurs or when any change occurs. The PCINT is cleared by reset.

PCINT

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED				EDM4	EDM5	EDM6	EDM7	EDM8	EDM9	EDM10	EDM11	EDM12	EDM13	EDM14	EDM15
RESET	0				0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W				R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x968															

Bits 0–3—Reserved.
These bits are reserved and should be set to 0.

EDM4–EDM15—Edge Detect Mode for Lines 4–15
The corresponding port C line asserts an interrupt request.

0 = Any change on PC*x* generates an interrupt request.
1 = High-to low change on PC*x* generates an interrupt request.

16.14.10 Port D Pin Functionality

The 13 port D pins are independently configured as general-purpose I/O pins if the corresponding port D pin assignment register (PDPAR) is cleared. They are configured as dedicated on-chip peripheral pins if the corresponding PDPAR bit is set.

As a general-purpose I/O pin, the signal direction is determined by the corresponding control bit in the port D data direction register (PDDIR). The port I/O pin is configured as an input if the corresponding PDDIR is cleared and it is configured as an output if the corresponding PDDIR is set. All PDPAR and PDDIR pins are cleared at total system reset, which configures all port D pins as general-purpose input pins. Refer to Table 16-44 for all the port D pin options' default descriptions.

Table 16-44. Port D Pin Assignment

SIGNAL	PDPAR = 0	PIN FUNCTION		INPUT TO ON-CHIP PERIPHERALS
		PDPAR = 1	PDDIR=0	
PD15	PORT D15	LD8	VD7	
PD14	PORT D14	LD7	VD6	
PD13	PORT D13	LD6	VD5	
PD12	PORT D12	LD5	VD4	
PD11	PORT D11	LD4	VD3	
PD10	PORT D10	LD3	VD2	
PD9	PORT D9	LD2	VD1	
PD8	PORT D8	LD1	VD0	
PD7	PORT D7	LD0	FIELD	
PD6	PORT D6	LCD_AC/$\overline{OE}$	BLANK	
PD5	PORT D5	FRAME/VSYNC	VV	
PD4	PORT D4	LOAD/HSYNC	VH	
PD3	PORT D3	SHIFT/CLK	VCLK	VCC

16.14.11 Port D Registers

Port D has three 16-bit, memory-mapped, read/write control registers.

16.14.11.1 PORT D DATA REGISTER. A read of the port D data (PDDAT) register returns the data on the pins, regardless of whether the pins are an input or an output. This allows output conflicts to be found on the pins by comparing the written data with the data on the pins. A write to the PDDIR is latched, and if that bit in the PDDIR is configured as an output, the value latched for that bit will be driven onto its respective pin. PDDAT can be read or written at any time. PDDAT is not initialized and is undefined by reset.

PDDAT

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED			D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
RESET	—			—	—	—	—	—	—	—	—	—	—	—	—	—
R/W	R/W			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x976															

NOTE: — = Undefined.

Bits 0–3—Reserved

These bits are reserved and should be set to 0.

D4–15—Data Pins 4–15

These bits contain data can be read or written from the port D pins.

16.14.11.2 PORT D DATA DIRECTION REGISTER. The port D data direction register (PDDIR) is cleared at system reset.

PDDIR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED			DR3	DR4	DR5	DR6	DR7	DR8	DR9	DR10	DR11	DR12	DR13	DR14	DR15
RESET	0			0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x970															

Bits 0–2—Reserved

These bits are reserved and should be set to 0.

DR3–DR14—Data Direction Pins 3–14

0 = The corresponding pin is an input.
1 = The corresponding pin is an output.

16.14.11.3 PORT D PIN ASSIGNMENT REGISTER. The port D pin assignment register (PDPAR) is cleared at system reset.

PDPAR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED			DD3	DD4	DD5	DD6	DD7	DD8	DD9	DD10	DD11	DD12	DD13	DD14	DD15
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x972															

Bits 0–2—Reserved
These bits are reserved and should be set to 0.

DD3–DD15—Dedicated Function

0 = General-purpose I/O. The peripheral functions of the pin are not used.
1 = Dedicated peripheral function. The pin is used by the internal module. The on-chip peripheral function to which it is dedicated can be determined by other bits such as those in the PDDIR.

16.15 THE CPM INTERRUPT CONTROLLER

The communication processor module's interrupt controller (CPIC) is the focal point for all interrupts associated with the communication processor module and it accepts and prioritizes the internal and external interrupt requests from the CPM blocks. It is also responsible for generating a vector during the core interrupt acknowledge cycle.

The CPM interrupt controller receives interrupts from such internal sources as the USB, SCC2, SMCs, SPI, I^2C, general-purpose timers, and port C parallel I/O pins. The CPM interrupt controller allows you to mask each interrupt source. When multiple events within a sub-block of the communication processor module cause the interrupt, each event is maskable in that sub-block.

All CPM sub-block interrupt sources are prioritized and bits are set in the CPM interrupt pending register (CIPR) where all interrupt sources are assigned one programmable priority level (0–7) before the request for an interrupt is sent to the U-Bus. An overview of the MPC823 interrupt structure is illustrated in Figure 16-133. The lower half of the figure illustrates the CPM interrupt controller.

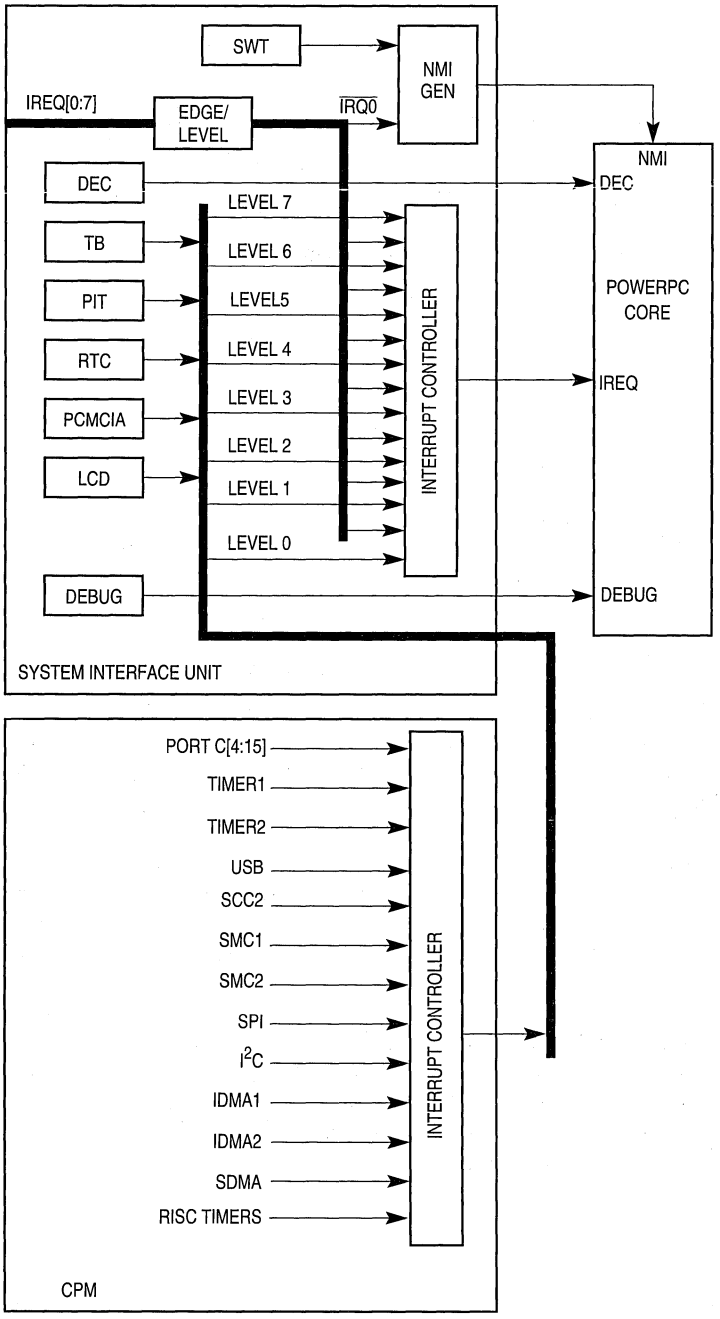


Figure 16-133. MPC821 Interrupt Structure

Within the CPM interrupt level, the sources are assigned a priority structure. On the MPC823, you have some flexibility with the relative priority of the interrupt sources.

Once an unmasked interrupt source is pending in the CIPR, the CPM interrupt controller sends an interrupt request to the U-Bus at level 0, 1, 2, 3, 4, 5, 6, or 7. The CPM interrupt controller then waits for the interrupt to be recognized. After the interrupt request is honored by the core, the core acknowledges the interrupt by setting the IACK bit in the CPM interrupt vector register. When the IACK bit is set, the CIVR is updated with a 5-bit vector corresponding to the sub-block with the highest current priority and the IACK is cleared after one clock cycle.

16.15.1 Features

The following is a list of the CPM interrupt controller's main features:

- Twenty-four Interrupt Sources (12 Internal and 12 External)
- Sources Can Be Assigned to a Programmable Interrupt Level
- Programmable Priority Between SCC2 and USB
- Two Priority Schemes for SCC2 and USB
- Programmable Highest Priority Request
- Fully Nested Interrupt Environment
- Unique Vector Number for Each Interrupt Source

16.15.2 CPM Interrupt Source Priorities

The CPM interrupt controller has 24 interrupt sources that assert a programmable interrupt request level to the core and the priority between these sources is shown in Table 16-45. There is some flexibility in the relative ordering of the interrupts in the table, but, in general, the relative priorities are fixed in the descending order shown. An interrupt from the parallel I/O signal PC15 has the highest priority and an interrupt from the parallel I/O signal PC4 has the lowest. A single interrupt priority number is associated with each table entry.

Notice the lack of SDMA interrupt sources. SDMA-related interrupts are reported through each individual USB, SCC2, SMC, SPI, or I^2C channel. The only true SDMA interrupt source is the SDMA channel bus error entry that is reported when a bus error occurs during an SDMA access. There are two ways to add flexibility to the table of CPM interrupt priorities (the USB/SCC2 relative priority option or the highest priority option).

16.15.2.1 USB AND SCC2 RELATIVE PRIORITY. The relative priority between the USB and SCC2 is programmable and can be dynamically changed. In Table 16-45 there is no entry for USB and SCC2, but rather there are entries for SCCa, SCCb, SCCc, and SCCd because each one of them can be mapped to any of these locations. This is programmed in the CICR and can be dynamically changed. You can use this on-the-fly capability to implement a rotating priority.

In addition, there are two ways to group the locations of the SCCa, SCCb, SCCc, and SCCd entries—group and spread. In the group scheme, the USB and SCC2 are all grouped together at the top of the priority table, ahead of most of the other CPM interrupt sources. This scheme is ideal for applications where USB and SCC2 function at a very high data rate and interrupt latency is very important. In the spread scheme, the USB and SCC2 priorities are spread over the table so that other sources can have lower interrupt latencies than the USB and SCC2. This scheme is also programmed in the CICR, but it cannot be dynamically modified.

16.15.2.2 HIGHEST PRIORITY INTERRUPT. In addition to the USB and SCC2 relative priority option, you can choose one interrupt source to be of highest priority. This highest priority interrupt is still within the same interrupt level as the rest of the CPIC interrupts, but is serviced prior to any other interrupt in the table. If the highest priority feature is not used, select PC15 to be the highest priority interrupt and no modifications to the standard interrupt priority order will be made. This highest priority source is dynamically programmable in the CICR and it allows you to change a normally low priority source into a high priority source for a certain period of time.

Table 16-45. Prioritization of CPM Interrupt Sources

NUMBER	PRIORITY LEVEL	INTERRUPT SOURCE DESCRIPTION	MULTIPLE EVENTS
1F	Highest	Parallel I/O–PC15	No
1E		SCCa (Grouped and Spread)	Yes
1D		SCCb (Grouped)	Yes
1C		SCCc (Grouped)	Yes
1B		SCCd (Grouped)	Yes
1A		Parallel I/O–PC14	No
19		Timer 1	Yes
18		Parallel I/O–PC13	No
17		Parallel I/O–PC12	No
16		SDMA Channel Bus Error	Yes
15		IDMA1	Yes
14		IDMA2	Yes
13		SCCb (Spread)	Yes
12		Timer 2	Yes
11		RISC Timer Table	Yes
10		I^2C	Yes
F		Parallel I/O–PC11	No
E		Parallel I/O–PC10	No
D		SCCc (Spread)	Yes
C		Reserved	Yes
B		Parallel I/O–PC9	No
A		Parallel I/O–PC8	No
9		Parallel I/O–PC7	No
8		SCCd (Spread)	Yes
7		Reserved	Yes
6		Parallel I/O–PC6	No
5		SPI	Yes
4		SMC1	Yes
3		SMC2	Yes
2		Parallel I/O–PC5	No
1		Parallel I/O–PC4	No
0	Lowest	Reserved	—

16.15.2.3 NESTED INTERRUPTS. The CPM interrupt controller supports a fully nested interrupt environment that allows a high priority interrupt from another CPM source to suspend a lower priority interrupt service routine. This nesting is achieved by the CPM interrupt in-service register (CISR). The CPM interrupt controller prioritizes all interrupt sources based on their assigned priority level. The highest priority interrupt request is presented to the core for servicing and the core acknowledges the interrupt by setting the IACK bit in the CIVR.

After the IACK bit is set, the vector number that corresponds to this interrupt is made available to the core in the CIVR and the interrupt request is cleared. If there are remaining interrupt requests, they are then prioritized and another interrupt request can be presented to the core. Upon interrupt, the interrupt mask bit in the machine status register (MSR) is cleared to disable further interrupt requests until the software is ready to handle them. Refer to **Section 6.4.1.2.1 Machine State Register** for more information.

The CISR can be used to allow a higher priority interrupt within the same interrupt level to be presented to the core before a lower priority interrupt service is completed. Each bit in the CISR corresponds to a CPM interrupt source. When the core acknowledges the interrupt by setting the IACK bit of the CIVR, the bits in the CISR is set by the CPM interrupt controller for that interrupt source.

Setting the bit prevents any subsequent CPM interrupt requests at this priority level or lower, until the servicing of the current interrupt has completed and you clear the in-service bit. Pending interrupts for these sources are still set in the CPM interrupt controller during this time which means that, in the interrupt service routine for the CPM interrupts, you can enable the core interrupt mask to allow higher priority interrupts within this level to generate an interrupt request. This capability provides nesting of interrupt requests for CPM interrupt level sources.

16.15.3 Masking Interrupt Sources in the CPM

By programming the CPM interrupt mask register (CIMR), you can mask the CPM interrupts to prevent an interrupt request to the core. Each bit in the CIMR corresponds to one of the CPM interrupt sources. To enable an interrupt, write a 1 to the corresponding CIMR bit. When a masked CPM interrupt source has a pending interrupt request, the corresponding bit in the CIPR is still set, even though the interrupt is not generated to the core. By masking all interrupt sources in the CIMR, you can implement a polling interrupt servicing scheme for the CPM interrupts.

When a CPM interrupt source has multiple interrupting events, you can individually mask these events by programming a mask register within that block. Table 16-46 shows the interrupt sources that have multiple interrupting events and Figure 16-134 illustrates an example of how the masking occurs using the SCC2 as an example.

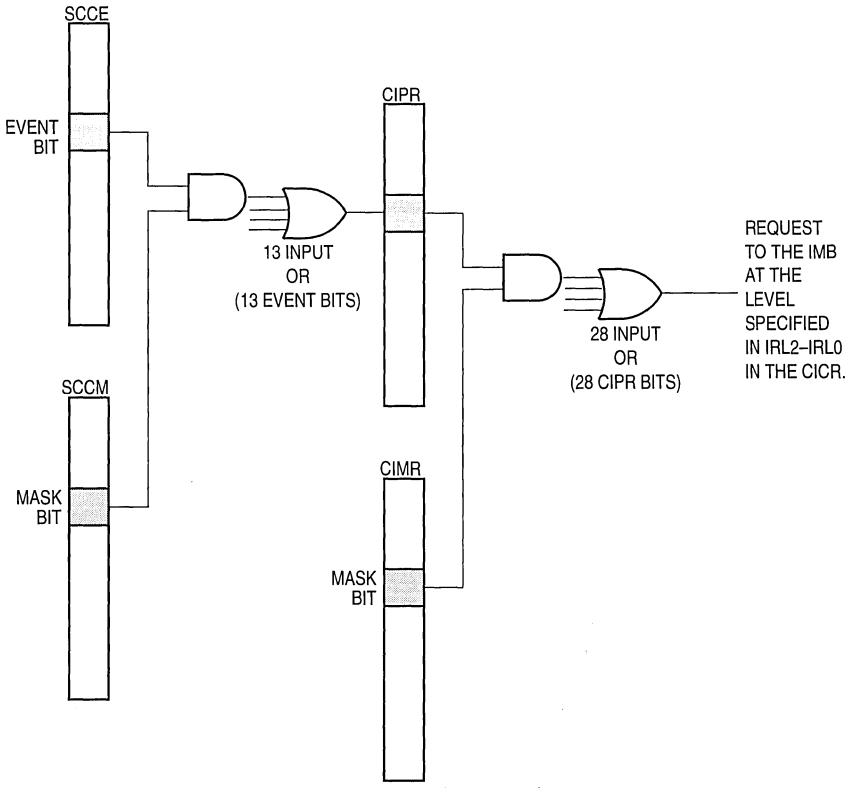


Figure 16-134. Interrupt Request Masking

16.15.4 Generating and Calculating an Interrupt Vector

All pending unmasked CPM interrupts are presented to the core in order of priority. The core responds to an interrupt request by setting the IACK bit in the CIVR. The interrupt vector that allows the core to locate the interrupt service routine is made available to the core by reading the CIVR. For CPM interrupts, the CPM interrupt controller passes an interrupt vector corresponding to the highest priority, unmasked, pending interrupt. The CPM interrupt controller encoding of the five low-order bits of the interrupt vector is provided in Table 16-46.

Table 16-46. Encoding the Interrupt Vector

INTERRUPT NUMBER	INTERRUPT SOURCE DESCRIPTION	INTERRUPT VECTOR
1F	Parallel I/O—PC15	11111
1E	USB	11110
1D	SCC2	11101
1C	Reserved	11100
1B	Reserved	11011
1A	Parallel I/O—PC14	11010
19	Timer 1	11001
18	Parallel I/O—PC13	11000
17	Parallel I/O—PC12	10111
16	SDMA Channel Bus Error	10110
15	IDMA1	10101
14	IDMA2	10100
13	Reserved	10011
12	Timer 2	10010
11	RISC Timer Table	10001
10	I^2C	10000
F	Parallel I/O—PC11	01111
E	Parallel I/O—PC10	01110
D	Reserved	01101
C	Reserved	01100
B	Parallel I/O—PC9	01011
A	Parallel I/O—PC8	01010
9	Parallel I/O—PC7	01001
8	Reserved	01000
7	Reserved	00111
6	Parallel I/O—PC6	00110
5	SPI	00101
4	SMC1	00100
3	SMC2	00011
2	Parallel I/O—PC5	00010
1	Parallel I/O—PC4	00001
0	Error	00000

The interrupt vector table is the same as the CPM interrupt priority table except for two differences. First, the USB and SCC2 vectors are fixed. They are not affected by the USB and SCC2 group mode, spread mode, or the relative priority order of the USB and SCC2. Second, an error vector is the last entry in this table. The error vector is issued by the communication processor module if it requested one but you masked it before it was serviced by the core and if there were no other pending interrupts. You should provide an error interrupt service routine, even if it is simply an **rfi** instruction.

16.15.5 Programming the CPM Interrupt Controller

There are four CPM interrupt controller registers:

- CPM interrupt configuration register—Defines the overall CPM interrupt attributes.
- CPM interrupt pending register—Indicates which CPM interrupt sources require interrupt service.
- CPM interrupt mask register—Allows you to prevent any CPM interrupt source from generating an interrupt request.
- CPM interrupt in-service register—Allows a fully nested environment capability for interrupt requests within the CPM interrupt level.

16.15.5.1 CPM INTERRUPT CONFIGURATION REGISTER. The 24-bit read/write CPM interrupt configuration register (CICR) defines the request level for the CPM interrupts, the priority between the USB and SCC2 and the highest priority interrupt.

CICR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED								SCdP		SCcP		SCbP		SCaP	
RESET	0								0		0		0		0	
R/W	R/W								R/W		R/W		R/W		R/W	
ADDR	(IMMR & 0xFFFF0000) + 0x940															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	IRL0	IRL1	IRL2	HP0	HP1	HP2	HP3	HP4	IEN	RESERVED						SPS
RESET	0	0	0	0	0	0	0	0	0	0						0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W						R/W
ADDR	(IMMR & 0xFFFF0000) + 0x942															

Bits 0–7 and 25–30—Reserved

These bits are reserved and should be set to 0.

SCdP—SCCd Priority Order

This field defines whether USB or SCC2 will assert its request in the SCCd priority position.

00 = USB will assert its request in the SCCd position.
01 = SCC2 will assert its request in the SCCd position.

SCcP—SCCc Priority Order

This field defines whether USB or SCC2 will assert its request in the SCCc priority position.

00 = USB will assert its request in the SCCc position.
01 = SCC2 will assert its request in the SCCc position.

SCbP—SCCb Priority Order

This field defines the serial communication controller that asserts its request in the SCCb priority position.

00 = USB asserts its request in the SCCb position.
01 = SCC2 asserts its request in the SCCb position.

SCaP—SCCa Priority Order

This field defines the serial communication controller that asserts its request in the SCCa priority position.

00 = USB asserts its request in the SCCa position.
01 = SCC2 asserts its request in the SCCa position.

Note: You should not program USB or SCC2 to more than one priority position (a, b, c, or d). These bits may be changed dynamically.

IRL—Interrupt Request Level

This field contains the priority request level of the interrupt from the communication processor module that is sent to the core. Level 0 indicates the highest priority interrupt and level 7 indicates the lowest. The IRL field is initialized to zero during reset. In most systems, value 0x4 is a good value to choose for these bits.

10 = Reserved.
11 = Reserved.

HP—Highest Priority

This field specifies the 5-bit interrupt number of the single CPM interrupt controller's interrupt source that is advanced to the highest priority in the table. These bits can be dynamically modified. To keep the original priority order intact, simply program these bits to 11111.

IEN—Interrupt Enable

This bit is a master enable for the CPM interrupts.

0 = CPM interrupts are disabled
1 = CPM interrupts are enabled

SPS—Spread Priority Scheme

This bit selects the relative USB or SCC2 priority scheme and cannot be changed dynamically.

0 = Grouped. The USB or SCC2 are grouped by priority at the top of the table.
1 = Spread. The USB or SCC2 are spread by priority in the table.

16.15.5.2 CPM INTERRUPT PENDING REGISTER. Each bit in the 32-bit read/write CPM interrupt pending register (CIPR) corresponds to a CPM interrupt source. When a CPM interrupt is received, the CPM interrupt controller sets the corresponding bit in the CIPR.

CIPR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PC15	USB	SCC2	RESERVED		PC14	TIMER 1	PC13	PC12	SDMA	IDMA1	IDMA2	RES	TIMER 2	R-TT	I2C
RESET	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x944															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	PC11	PC10	RESERVED		PC9	PC8	PC7	—	RES	PC6	SPI	SMC1	SMC2	PC5	PC4	RES
RESET	0	0	0		0	0	0	0		0	0	0	0	0	0	0
R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x946															

In a vectored interrupt scheme, the CPM interrupt controller clears the bit in the CIPR that corresponds to the current interrupt when the core acknowledges the interrupt. The core acknowledges the interrupt by setting the IACK bit in the CIVR. The vector number that corresponds to the CPM interrupt source is then available to the core in the CIVR. However, the CIPR bit is not cleared if an event register exists for that interrupt source. Event registers only exist for interrupt sources that have multiple source events. For example, the serial communication controller has multiple events that cause serial communication controller interrupts.

In a polled interrupt scheme, you must periodically read the CIPR. When a pending interrupt is handled, clear the corresponding bit in the CIPR. However, if an event register exists, clear the unmasked event register bits instead, thus causing the CIPR bit to be cleared. To clear a bit in the CIPR, write a 1 to that bit. Since you can only clear bits in this register, bits written as zeros are unaffected. The CIPR is cleared at reset.

Note: The USB or SCC2 CIPR bit positions are not changed according to the relative priority between USB or SCC2 (as determined by the SCxP and SPS bits in the CICR). Writing a zero to a bit in the CIPR has no effect.

16.15.5.3 CPM INTERRUPT MASK REGISTER. Each bit in the 32-bit read/write CPM interrupt mask register (CIMR) corresponds to a CPM interrupt source. You can mask an interrupt by clearing the corresponding bit in the CIMR and you can enable one by setting the corresponding bit in the CIMR. When a masked CPM interrupt occurs, the corresponding bit in the CIPR is still set, regardless of the CIMR bit, but no interrupt request is passed to the core.

If a CPM interrupt source is requesting interrupt service when you clear its corresponding bit in the CIMR, the request stops. If you set its bit in the CIMR later, a previously pending interrupt request is processed by the core, according to its assigned priority. You can read the CIMR at any time and it is cleared by reset.

CIMR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PC15	USB	SCC2	RESERVED		PC14	TIMER 1	PC13	PC12	SDMA	IDMA1	IDMA2	RES	TIMER 2	R–TT	I2C
RESET	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x948															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	PC11	PC10	RESERVED		PC9	PC8	PC7	RESERVED		PC6	SPI	SMC1	SMC2	PC5	PC4	RES
RESET	0	0	0		0	0	0	0		0	0	0	0	0	0	0
R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x94A															

Note: The USB or SCC2 CIMR bit positions are unaffected by the relative priority between USB or SCC2. To clear bits that were set by multiple interrupt events, you must clear all the unmasked events in the corresponding event register. If a bit in the CIMR is masked at the same time that the corresponding CIPR bit causes an interrupt request to the core, then the interrupt is not processed, but the error vector is issued if the interrupt acknowledge cycle occurs with no other CPM interrupts pending. Thus, you should always include an error vector routine, even if it just contains the **rfi** instruction. The error vector cannot be masked.

16.15.5.4 CPM INTERRUPT IN-SERVICE REGISTER. Each bit in the 32-bit read/write CPM interrupt in-service register (CISR) corresponds to a CPM interrupt source. In a vectored interrupt environment, the CPM interrupt controller sets the CISR bit when the core acknowledges the interrupt by setting the IACK bit in the CPM interrupt vector register. Your interrupt service routine must clear this bit after servicing is complete. If an event register exists for this peripheral, its bits would normally be cleared as well. To clear a bit in the CISR, write a 1 to that bit. Since you can only clear bits in this register, bits written as zeros will not be affected. The CISR is cleared by reset.

CISR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PC15	USB	SCC2	RESERVED		PC14	TIMER 1	PC13	PC12	SDMA	IDMA1	IDMA2	RES	TIMER 2	R–TT	I2C
RESET	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x94C															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	PC11	PC10	RESERVED		PC9	PC8	PC7	RESERVED		PC6	SPI	SMC1	SMC2	PC5	PC4	RES
RESET	0	0	0		0	0	0	0		0	0	0	0	0	0	0
R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x94E															

You can read this register to determine the interrupt requests that are currently in progress for each CPM interrupt source. More than one bit in the CISR can be a 1 if higher priority CPM interrupts are allowed to interrupt lower priority level interrupts within the same CPM interrupt level. For example, the TIMER1 interrupt routine could interrupt the handling of the TIMER2 routine using a special nesting technique described earlier. During this time, you can see both the TIMER2 and the TIMER1 bits simultaneously set in the CISR.

Note: The USB or SCC2 CISR bit positions are not affected by the relative priority between USB or SCC2. If the error vector is taken, no bit in the CISR is set. All undefined bits in the CISR return zeros when read. You can control the extent to which CPM interrupts can interrupt other CPM interrupts by selectively clearing the CISR. A new interrupt is processed if it has a higher priority than the higher priority interrupt having its CISR bit set. Thus, if an interrupt routine sets the interrupt mask bit in the core and also clears its CISR bit at the beginning of the interrupt routine, a lower priority interrupt can interrupt the higher one, as long as the lower priority interrupt is of higher priority than any other CISR bits that are currently set.

16.15.5.5 CPM INTERRUPT VECTOR REGISTER. The CPM interrupt vector register (CIVR) is a 16-bit register. Bits 0-4 of the register contain the interrupt vector number. To update the register with the current interrupt vector number, the core should set the IACK bit. The bit is cleared after one clock cycle. The register can be read at any time.

CIVR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	VECTOR NUMBER					RESERVED										IACK
RESET	0					0										0
R/W	R/W					R/W										R/W
ADDR	(IMMR & 0xFFFF0000) + 0x930															

16.15.6 Interrupt Handling Examples

You can use the following examples to learn how to properly handle CPM interrupts. However, there is nothing here about nesting interrupts within the CPM interrupt level.

16.15.6.1 PC6 INTERRUPT HANDLER EXAMPLE. In this example, the CPM interrupt controller hardware clears the PC6 bit in the CIPR during the interrupt acknowledge cycle. The following steps show you how to handle an interrupt source without multiple events.

1. Set the IACK bit in the CIVR.
2. Read the vector to access interrupt handler.
3. Handle the event associated with a change in the state of the PC6 pin.
4. Clear the PC6 bit in the CISR.
5. Execute the **rfi** instruction.

16.15.6.2 USB INTERRUPT HANDLER EXAMPLE. In this example, the USB bit in the CIPR remains set as long as one or more unmasked event bits remain in the USBE register. This is an example of a handler for an interrupt source with multiple events. Notice that the bit in the CIPR does not need to be cleared by the handler, but the bit in the CISR does.

1. Set the IACK bit in the CIVR.
2. Read the vector to access interrupt handler.
3. Immediately read the USBE register into a temporary location.
4. Decide which events in the USBE will be handled in this handler and clear those bits as soon as possible. USBE bits are cleared by writing ones.
5. Handle the events in the USB RX or TX buffer descriptor tables.
6. Clear the USB bit in the CISR.
7. Execute the **rfi** instruction. If any unmasked bits in the USBE remain at this time (either not cleared by the software or set by the MPC823 during the execution of this handler), this interrupt source is once again made pending immediately following the **rfi** instruction.

SECTION 17
PCMCIA INTERFACE

The PCMCIA host adapter module provides all control logic for a PCMCIA interface. Only the analog power switching logic and buffering needs to be provided externally. The PCMCIA host supports one PCMCIA socket.

17.1 FEATURES

The following list summarizes the main features of the PCMCIA interface:

- A Host Adapter Interface Fully Compliant with the PCMCIA Standard, Release 2.1+ (PC Card -16).
- Supports One PCMCIA Socket, Requiring Only External Buffering and Analog Switching Logic. The Socket Is Referred to As Slot B to Maintain Compatibility with the MPC821 Microprocessor.
- Supports Eight Memory or I/O Windows.
- Provides Eight General-Purpose I/O Pins When the PCMCIA Controller Is Not In Operation.
- Provides Two General-Purpose Output-Only Pins When the PCMCIA Controller Is Not In Operation.

17.2 SYSTEM CONFIGURATION

The PCMCIA host adapter interface module can control one PCMCIA socket, which is illustrated in Figure 17-1. In this system configuration, you must accomplish electrical isolation between the sockets and system bus using external buffers and bus transceivers. These buffers also provide the voltage conversion needed from the MPC823's 3.3V to 5V cards. They should be powered by the card V_{cc}. Since the MPC823 is 5V friendly and will accept 5V inputs while generating 3.3V outputs, no conversion is needed for inputs.

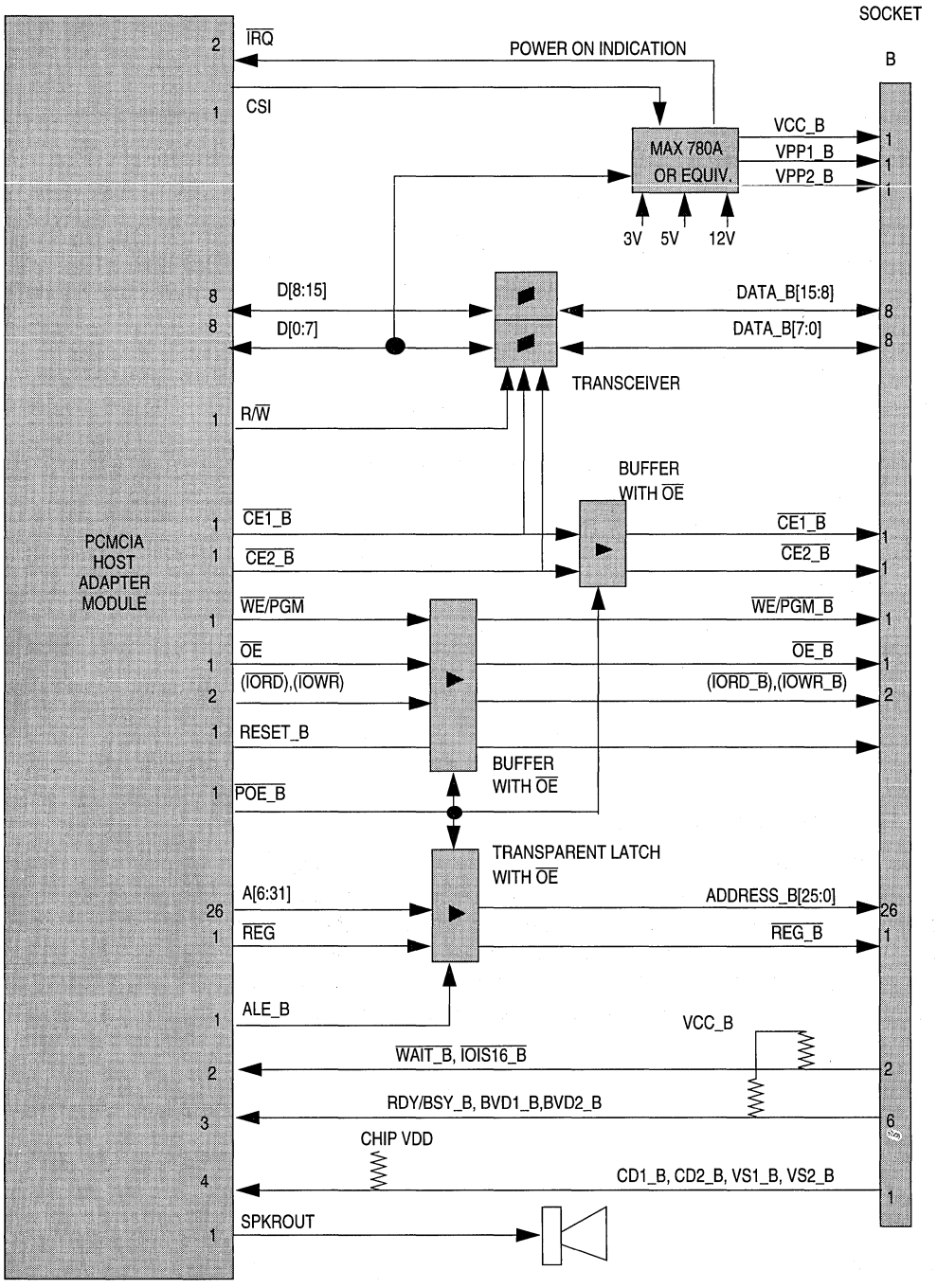


Figure 17-1. System with One PCMCIA Socket

17.3 PCMCIA SIGNALS

The PCMCIA module consists of the cycle control, input port, output port, and various other signals.

17.3.1 The PCMCIA Cycle Control Signals

The following signals are used for I/O accesses to the PCMCIA card:

- Address Bus (A[6:31])—Output. These signals should be buffered to generate the socket's A25 through A0 signals, which are address bus output lines that allow direct addressing of up to 64M of memory on each PCMCIA card. Signal A6 is the most-significant bit and A31 is the least-significant bit.
- Attribute Memory Select ($\overline{REG}$)—Output. When this signal is asserted during a PCMCIA access, card access is limited to attribute memory when a memory access occurs ($\overline{WE}$ or $\overline{OE}$ are asserted), and to I/O ports when an I/O access occurs ($\overline{IORD}$ or $\overline{IOWR}$ are asserted). On accesses with $\overline{REG}$ asserted, accesses to common memory or DMA devices are blocked. When no PCMCIA access is performed this signal is TSIZ0.

  Card Enables ($\overline{CE1\_B}$, $\overline{CE2\_B}$)—Output. When a PCMCIA access is performed, the $\overline{CE1}$ and $\overline{CE2}$ signals are card enable output signals. The $\overline{CE1}$ signal enables even bytes and $\overline{CE2}$ enables odd bytes.

  The $\overline{CE}$ signals can be configured to duplicate the values of the A[22:23] signals. At the end of the PCMCIA access these lines will be always negated. See Table 17-1 for details. This feature can be used to access devices supporting IDE/ATA protocols.

Table 17-1. Card Enable as Driven by the MPC823

PRS FIELD	A22	A23	PORT SIZE	ACCESS SIZE	MPC823: A31 (SLOT: A0)	CE2	CE1
PRS ≠ '110'	X	X	8 bits	16 bit (only even)	0	1	0
				8 bit odd	1	1	0
				8 bits even	0	1	0
			16 bits	16 bit (only even)	0	0	0
				8 bit odd	1	0	1
				8 bits even	0	1	0
				No access	X	1	1
PRS = '110'	0	0	X	X	X	0	0
	0	1	X	X	X	0	1
	1	0	X	X	X	1	0
	1	1	X	X	X	1	1

- Data Bus (D[0:15])—Bidirectional. Signals D0 through D15 constitute the bidirectional data bus. The most-significant bit is D0. Significance decreases downward to D15.
- Extend Bus Cycle ($\overline{\text{WAIT\_B}}$)—Input. This signal is asserted by the PC card to delay completion of the pending memory or I/O cycle.
- External Transceiver Direction (R/$\overline{\text{W}}$)—Output. This signal is part of the MPC823 bus. It is asserted or driven high during any read cycles of the MPC823 and negated or driven low during write cycles. It is used in the PCMCIA interface to control the direction of the data bus transceivers.
- I/O Read ($\overline{\text{IORD\_B}}$)—Output. During PCMCIA accesses, this signal is asserted in conjunction with $\overline{\text{REG\_B}}$. It is used to read data from the PC card's I/O space. $\overline{\text{IORD\_B}}$ is valid only when the $\overline{\text{REG\_B}}$ and at least one of $\overline{\text{CE1\_B}}$ and $\overline{\text{CE2\_B}}$ signals is asserted.
- I/O Write ($\overline{\text{IOWR\_B}}$)—Output. During PCMCIA accesses, this signal is asserted in conjunction with $\overline{\text{REG\_B}}$. It is used to latch data into the PC card's I/O space. $\overline{\text{IOWR\_B}}$ is valid only when the $\overline{\text{REG\_B}}$ and at least one of $\overline{\text{CE1\_B}}$ and $\overline{\text{CE2\_B}}$ signals is asserted.
- Output Enable ($\overline{\text{OE\_B}}$)—Output. During PCMCIA accesses, the $\overline{\text{OE\_B}}$ signal is used to drive memory read data from a PC card into a PCMCIA socket.
- Write Enable/Program ($\overline{\text{WE\_B}}$)—Output. During PCMCIA accesses, the $\overline{\text{WE\_B}}$ signal is used to latch memory write data to the PC card in a PCMCIA socket. This signal can also be used as the programming strobe for PC cards employing programmable memory technologies.
- Address Latch Enable (ALE_B)—Output. This strobe signal controls the external buffers of the Address and $\overline{\text{REG}}$ signals.
- I/O Port Is 16 Bits ($\overline{\text{IOIS16\_B}}$)—Input. When the card and its socket are programmed for I/O interface operation, this signal is used as $\overline{\text{IOIS16\_B}}$ and must be asserted by the card when the address on the bus corresponds to an address on the PC card and the I/O port being addressed is capable of 16-bit accesses. If the I/O region in which the address resides is programmed as 8 bits wide, then the $\overline{\text{IOIS16\_B}}$ signal is ignored.

17.3.2 The PCMCIA Input Port Signals

The MPC823 provides synchronization, transition detection, optional interrupt generation and a means for the software to read the signal state. This function is not necessarily specific to PCMCIA and the signals can be used as a general-purpose input port with edge detection and interrupt capability.

The following signals are used by a PCMCIA slot to indicate the status of the card. They appear on the IP_B[0:7] pins, which you can access through bits 16-23 of the PIPR when you are not operating the PCMCIA controller. All these signals are symmetrical except for IP_B[7], which has extended edge detection capability, and IP_B[2] that serves as $\overline{\text{IOIS16\_B}}$ cycle control signals for PCMCIA cycles.

- Voltage Sense ($\overline{\text{VS1\_B}}$, $\overline{\text{VS2\_B}}$)—Input. These signals are used as VS1 and VS2 and are generated by PC cards. They notify the socket of the card's V_{CC} requirement. These signals are connected to the IP_B[0:1] pins.
- Write Protect (WP)—Input. When the card and its socket are programmed for memory interface operation, this signal is used as WP and reflects the status of the write protect switch on the PC card. It must be asserted by the PC card when the switch is enabled and negated when the switch is disabled. For a PC card without a switch, this signal must be connected to ground if the PC card can be written and connected to system V_{CC} if the PC card is permanently write-protected. This signal is connected to the IP_B[2] pin.
- Card Detect ($\overline{\text{CD1\_B}}$, $\overline{\text{CD2\_B}}$)—Input. These signals ensure that a card has been inserted properly. They must be connected to ground internally on the PC card and they will be forced low whenever a card is placed in the socket. These signals must be pulled up to system V_{CC} to allow card detection to function while the card socket is powered down. These signals are connected to the IP_B4 and IP_B3 pins, respectively.
- Battery Voltage Detect (BVD1_B, BVD2_B)—Input. When the card and its socket are programmed for memory interface operation, these signals are used as BVD1_B and BVD2_B and are generated by PC cards having an onboard battery. They report the battery's condition. Both BVD1_B and BVD2_B must be held asserted when the battery is in good condition. Negating BVD2_B while keeping BVD1_B asserted indicates the battery is in a warning condition and should be replaced, although data integrity on the card is still assured. Negating BVD1_B indicates that the battery is no longer serviceable and data is lost, regardless of the state of BVD2_B. These signals are connected to the IP_B6 and IP_B5 pins, respectively.
- Status Change ($\overline{\text{STSCHG}}$)—Input. When the card and its socket are programmed for I/O Interface operation, the BVD1_B signal is used as $\overline{\text{STSCHG}}$, it is generated by the I/O PC card. The $\overline{\text{STSCHG}}$ signal must be held negated when either or both the Signal on Change bit and Changed bit in the card status register on the PC card are set to zero. The $\overline{\text{STSCHG}}$ signal must be asserted when both bits are set to one.

- Speaker ($\overline{\text{SPKR}}$)—Input. When the card and its socket are programmed for I/O interface operation, the BVD2_B signal is used as $\overline{\text{SPKR}}$ and it is generated by the I/O PC cards. The $\overline{\text{SPKR}}$ signal must be used to provide the socket's single amplitude (digital) audio waveform to the system. The $\overline{\text{SPKR}}$ signal is routed out the SPKROUT signal if the card and its socket are programmed for I/O interface operation.
- Ready/Busy/Interrupt Request (RDY/BSY_B/$\overline{\text{IREQ\_B}}$)—Input. When the card and its socket are programmed for memory interface operation, this signal is used as RDY/BSY_B and must be asserted low by a PC card to indicate that the PC card is busy processing a previous **WRITE** command. When the card and its socket are programmed for I/O operation, this signal is used as $\overline{\text{IREQ\_B}}$ and must be asserted low to generate an interrupt. This signal must be set high when no interrupt is requested. This signal is connected to the IP_B[7] pin.

17.3.3 The PCMCIA Output Port Signals

The following signals are used by a PCMCIA slot to control the $\overline{\text{RESET}}$ signal to the card and the output enable of the buffers to the card. The MPC823 provides a way for the software to control the output signal state. This function is not necessarily specific to the PCMCIA interface and these signals may be used by a system as a general-purpose output port. These signals appear on the OP[2:3] pins. The OP[2:3] pins, when not operating the PCMCIA controller, can be accessed as general-purpose outputs through the CBOE and CBRESET bits of the PGCRB register.

- Card Reset (RESET_B)—Output. This signal is provided to clear the card configuration option register residing on the card, thus placing the card in its default (memory-only interface) state and initiating the beginning of any additional card initialization. RESET_B is connected to the OP3 pin.
- PCMCIA Buffers Output Enable ($\overline{\text{POE\_B}}$)—Output. This line is an output port line reflecting the value of the CBOE bit in the PCMCIA interface power control register. It should be used to three-state the address and strobe pins addressing the slot. $\overline{\text{POE\_B}}$ is connected to the OP2 pin.

17.3.4 Other PCMCIA Signals

The following special function signals can be used by the PCMCIA socket. Their function is not necessarily specific to PCMCIA and these signals can be used by a system for other functions.

- Power Is On ($\overline{\text{IRQ}}$)—Input. One $\overline{\text{IRQ}}$ signal provided for general-purpose interrupt requests may be used by the card power supply circuitry to notify the MPC823 processor when the power supply to the card has reached the required voltage.
- Speaker out (SPKROUT)—Output. This signal is used to provide a digital audio waveform that is to be driven to the system's speaker. It is generated from the SPKR input port signal.

17.4 PCMCIA OPERATION

17.4.1 Memory-Only Cards

Table 17-2 shows a worst case example of host programming.

Table 17-2. Host Programming for Memory Cards

MEMORY ACCESS TIME	600NS			200NS			150NS			100NS		
	STP	LNG	HLD	STP	LNG	HLD	STP	LNG	HLD	STP	LNG	HLD
CLK CYCLE	100	300	150	30	120	90	20	80	75	15	60	50
20ns	6	24	8	2	8	5	2	6	4	1	4	3
30ns	4	16	5	2	5	3	1	4	3	1	3	2
40ns	3	12	4	1	4	3	1	3	2	1	2	2
62ns	2	8	3	1	2	2	1	2	2	1	1	1
83ns	2	6	2	1	2	2	1	1	1	1	1	1

Note: Because the minimum hold time is one clock, the real access time is, access time plus one clock. Hold time and setup time HLD and STP, in this table, are the read or write worst case. The worst case hold time is "data disable from $\overline{OE}$. The worst case setup time is address to strobe. Length (LNG) in this table is the minimum strobe time.

Table 17-2 assumes you are not using the $\overline{WAIT\_B}$ signal. If you are using the $\overline{WAIT\_B}$ signal, then the minimum strobe time is at least 35ns + 1 system clock.

17.4.2 I/O Cards

Table 17-3 shows a worst case example of programming PCMCIA host for I/O cycle.

Table 17-3. Host Programming For I/O Cards

	STP(1)	LNG	HLD
20ns	4	8	2
30ns	3	6	1
40ns	3	4	1
62ns	2	3	1
83ns	2	2	1

Setup time worst case is for write, in which case, setup = data_setup_before_IORD +1 system clock.

17.4.3 Interrupts

Each input from the PCMCIA card to the host (BVD,CD,RDY, and VS) is sampled in the PCMCIA interface input pins register (PIPR) and any change to these bits is reported in the PCMCIA interface status change register (PSCR). The contents of the PSCR is AND'ed with the PCMCIA interface enable register (PER) to generate a PCMCIA interface interrupt. You can program the interrupt level for the exception that is generated. The PCMCIA interface can generate an additional interrupt for the RDY/$\overline{\text{IRQ}}$ signal. This interrupt can be generated for level (low or high) and for change (fall or rise) of the input signal.

17.4.4 Power Control

You can perform a write cycle using one of memory controller's $\overline{\text{CS}}$ signals, so that you can operate an external device to provide a regulated source voltage to the PCMCIA slot. There are a number of such devices available. However, auto-power control is not supported.

17.4.5 Reset and Three-State Control

You can write to the right bit in the PCMCIA general interface control register B (PGCRB) and cause the PCMCIA card to be reset or to disable the output drive of the external latches.

17.4.6 DMA

The MPC823's DMA module with the CPM microcode provides two independent DMA channels. The PCMCIA module can be programmed to generate control for an I/O device implemented as a PCMCIA card to act upon a DMA transfer. You can use the PRS field in the appropriate POR to program any window to be a DMA window. The PCMCIA controller supplies the signaling for the socket. DMA to or from the PCMCIA interface is done through dual-access DMA transfers. DMA requests can be supplied through the $\overline{\text{SPKR}}$, $\overline{\text{IOIS16\_B}}$ or $\overline{\text{INPACK}}$ signals. To support the DMA function, the slot's $\overline{\text{INPACK}}$ should be connected to $\overline{\text{DREQ2}}$. The actual source used for a DMA request is programmed in the CBDREQ[0:1] field of the PGCRB register. If the internal DMA request is enabled, then port C should be programmed to not $\overline{\text{DREQ2}}$. When the internal DMA request is disabled, then the DMA request is assumed to be $\overline{\text{DREQ2}}$. In this case, you should program port C so that the PC14 pin is $\overline{\text{DREQ2}}$.

Note: The PCMCIA controller will monitor the $\overline{\text{SDACK2}}$ signal internally to meet the IDMA handshaking protocol. Therefore, you do not have to monitor this signal for your PCMCIA design.

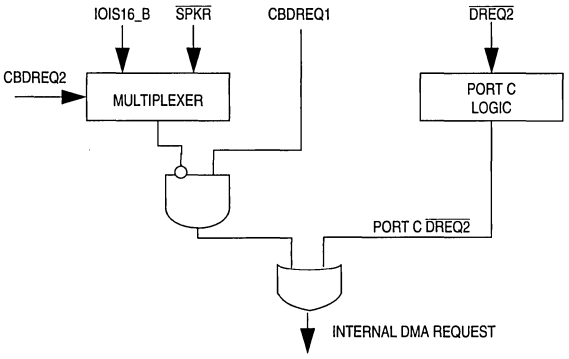


Figure 17-2. Internal DMA Request Logic

17.5 PCMCIA PROGRAMMING MODEL

The following section describes the PCMCIA interface programming model. All registers are memory-mapped within the internal control register area. The following registers are used to control the PCMCIA interface.

17.5.1 PCMCIA Interface Input Pins Register

The PCMCIA interface input pins register (PIPR) is used to sample the PCMCIA input port signals. When the PCMCIA controller is not operating, bits 16-23 of the PIPR can be used to read from and write to the IP_B[0:7] pins as general-purpose I/O pins.

PIPR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0xF0															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	CBVS1	CBVS2	CBWP	CBCD2	CBCD1	CBBVD 2	CBBVD 1	CBRDY	RESERVED							
RESET	—	—	—	—	—	—	—	—	—							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W							
ADDR	(IMMR & 0xFFFF0000) + 0xF0															

NOTE: — = Undefined.

Bits 0–15—Reserved

These bits are reserved and should be set to 0.

$\overline{\text{CBVS1}}$—Card B Voltage Sense 1

0 = Card B cannot operate at 3.3V.
1 = Card B can operate at 3.3V.

$\overline{\text{CBVS2}}$—Card B Voltage Sense 2

This bit is reserved by the PCMCIA controller for a secondary operating voltage. It should normally be set to 1 by Card B.

$\overline{\text{CBWP}}$—Card B Write-Protect

0 = Card B is not write-protected. It cannot be written to.
1 = Card B is write-protected.

$\overline{\text{CBCD2}}$—Card B Card Detect 2

0 = Card B is fully connected in socket.
1 = Card B is not properly connected in socket.

$\overline{\text{CBCD1}}$—Card B Detect 1

0 = Card B is fully connected in socket.
1 = Card B is not properly connected in socket.

CBBVD2—Card B Battery Voltage 2/$\overline{\text{SPKR}}$

If Card B and its socket are configured for I/O interface operation, this bit reflects the value of the $\overline{\text{SPKR}}$ signal. If Card B and its socket are configured for memory interface operation and Card B has an onboard memory, CBBVD2 is:

X0 = Card B battery is no longer serviceable and data is lost.
01 = Card B battery is in a warning condition and should be replaced. Card B data integrity is still assured.
11 = Card B battery is in good condition.

CBBVD1—Card B Battery Voltage 2/$\overline{\text{STSCHG}}$

If Card B and its socket are configured for I/O interface operation, this bit reflects the value of the $\overline{\text{STSCHG}}$ signal. If Card B and its socket are configured for memory interface operation and Card B has an onboard memory, CBBVD1 is:

X0 = Card B battery is no longer serviceable and data is lost.
01 = Card B battery is in a warning condition and should be replaced. Card B data integrity is still assured.
11 = Card B battery is in good condition.

CBRDY—Card B RDY/BSY_B/$\overline{\text{IREQ\_B}}$/$\overline{\text{IRQ}}$

If Card B and its socket are configured for memory interface operation, CBRDY is:

0 = Card B is busy.
1 = Card B is ready to accept a new data transfer operation.

If Card B and its socket are configured for I/O interface operation or if the card's power supply circuitry is using the $\overline{IRQ}$ signal, CBRDY is:

0 = Card B is requesting an interrupt.
1 = Card B is not requesting an interrupt.

Bits 24–31—Reserved

These bits are reserved and should be set to 0.

17.5.2 PCMCIA Interface Status Change Register

The PCMCIA interface status change register (PSCR) records changes in the state of the PCMCIA input port signals. The nonreserved bits in this register are reset by writing ones to them. Writing zero has no effect.

PSCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0xE8															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	CBVS1_C	CBVS2_C	CBWP_C	CBCD2_C	CBCD1_C	CBBVD2_C	CBBVD1_C	RES	CBRDY_L	CBRDY_H	CBRDY_R	CBRDY_F	RESERVED			
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0xE8															

NOTE: — = Undefined.

Bits 0–15—Reserved

These bits are reserved and should be set to 0

CBVS1_C—Card B Voltage Sense 1 Change

0 = Signal is changed.
1 = Signal is unchanged.

CBVS2_C—Card B Voltage Sense 2 Change

0 = Signal is changed.
1 = Signal is unchanged.

CBWP_C—Card B Write-Protect Change

0 = Signal is changed.
1 = Signal is unchanged.

CBCD2_C—Card B Card Detect 2 Change

0 = Signal is changed.
1 = Signal is unchanged.

CBCD1_C—Card B Card Detect 1 Change

0 = Signal is changed.
1 = Signal is unchanged.

CBBVD2_C—Card B Battery Voltage 2/$\overline{\text{SPKR}}$ Change

0 = Signal is changed.
1 = Signal is unchanged.

CBBVD1_C—Card B Battery Voltage 1/$\overline{\text{STSCHG}}$ Change

0 = Signal is changed.
1 = Signal is unchanged.

Bit 23—Reserved

This bit is reserved and should be set to 0.

CBRDY_L—Card B RDY/$\overline{\text{IRQ}}$ Low

0 = Signal is changed.
1 = Signal is unchanged.

CBRDY_H—Card B RDY/$\overline{\text{IRQ}}$ High

0 = Signal is changed.
1 = Signal is unchanged.

CBRDY_R—Card B RDY/$\overline{\text{IRQ}}$ Rising Edge

0 = Signal is changed.
1 = Signal is unchanged.

CBRDY_F—Card B RDY/$\overline{\text{IRQ}}$ Falling Edge

0 = Signal is changed.
1 = Signal is unchanged.

Bits 28–31—Reserved

These bits are reserved and should be set to 0

Note: Writing logic one to each bit reset it's value (zero), except for bit 9 which is always set to one.

17.5.3 PCMCIA Interface Enable Register

The PCMCIA interface enable register (PER) acts as a mask for the various sources of a PCMCIA interrupt.

PER

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0xF8															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	CB_EVS1	CB_EVS2	CB_EWP	CB_ECD2	CB_ECD1	CB_EBVD2	CB_EBVD1	RES	CB_ERDY_L	CB_ERDY_H	CB_ERDY_R	CB_ERDY_F	RESERVED			
RESET	—	—	—	—	—	—	—	—	—	—	—	—	—			
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W			
ADDR	(IMMR & 0xFFFF0000) + 0xF8															

NOTE: — = Undefined.

Bits 0–15—Reserved

These bits are reserved and should be set to 0.

CB_EVS1—Card B Enable for Voltage Sense 1

0 = Disable interrupt on any change in the relevant pin.
1 = Enable interrupt on changes of the relevant pin.

CB_EVS2—Card B Enable for Voltage Sense 2

0 = Disable interrupt on any change in the relevant pin.
1 = Enable interrupt on changes of the relevant pin.

CB_EWP—Card B Enable for Write-Protect

0 = Disable interrupt on any change in the relevant pin.
1 = Enable interrupt on changes of the relevant pin.

CB_ECD2—Card B Enable for Card Detect 2

0 = Disable interrupt on any change in the relevant pin.
1 = Enable interrupt on changes of the relevant pin.

CB_ECD1—Card B Enable for Card Detect 1

0 = Disable interrupt on any change in the relevant pin.
1 = Enable interrupt on changes of the relevant pin.

CB_EBVD2—Card B Enable for Battery Voltage/$\overline{\text{SPKR}}$

0 = Disable interrupt on any change in the relevant pin.
1 = Enable interrupt on changes of the relevant pin.

CB_EBVD1—Card B Enable for Battery Voltage/$\overline{\text{STSCHG}}$

0 = Disable interrupt on any change in the relevant pin.
1 = Enable interrupt on changes of the relevant pin.

Bit 23—Reserved

This bit is reserved and should be set to 0.

CB_ERDY_L—Card B Enable for RDY/$\overline{\text{IRQ}}$ Low

0 = Disable interrupt on any change in the relevant pin.
1 = Enable interrupt on changes of the relevant pin.

CB_ERDY_H—Card B Enable for RDY/$\overline{\text{IRQ}}$ High

0 = Disable interrupt on any change in the relevant pin.
1 = Enable interrupt on changes of the relevant pin.

CB_ERDY_R—Card B Enable for RDY/$\overline{\text{IRQ}}$ Rising Edge

0 = Disable interrupt on any change in the relevant pin.
1 = Enable interrupt on changes of the relevant pin.

CB_ERDY_F—Card B Enable for RDY/$\overline{\text{IRQ}}$ Falling Edge

0 = Disable interrupt on any change in the relevant pin.
1 = Enable interrupt on changes of the relevant pin.

Bits 28–31—Reserved

These bits are reserved and should be set to 0.

17.5.4 PCMCIA Interface General Control Register B

The PCMCIA interface general control register B (PGCRB) provides control for the IREQ and STSCHG interrupt levels, This register also controls the PCMCIA output port signals. When the PCMCIA controller is not operating, the CBOE and CBRESET bits can be used to access the OP[2:3] pins as general-purpose output pins without configuring any other PCMCIA register.

PGCRB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	CBIRQLVL								CBSCHLVL							
RESET	—								—							
R/W	R/W								R/W							
ADDR	(IMMR & 0xFFFF0000) + 0xE4															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	CBDREQ		RESERVED						CBOE	CBRESET	RESERVED					
RESET	—		—						—	—	—					
R/W	R/W		R/W						R/W	R/W	R/W					
ADDR	(IMMR & 0xFFFF0000) + 0xE4															

NOTE: — = Undefined.

CBIRQLVL—Card B Interrupt Request Level

Only one bit of this field should be set at any given time.

CBSCHLVL—Card B $\overline{\text{STSCHG}}$ Level

Only one bit of this field should be set at any given time.

CBDREQ—Card B DMA Request

This field defines the pin to be used as the internal DMA request to IDMA channel 2.

0x = Disable internal DMA request from Slot B.
10 = Enable $\overline{\text{IOIS16\_B}}$ as internal DMA request for Slot B.
11 = Enable $\overline{\text{SPKR}}$ as internal DMA request for Slot B.

Note: If the PCMCIA controller is programmed to enable internal DMA, then the port C registers should be configured to NOT select $\overline{\text{DREQ2}}$. Otherwise, $\overline{\text{DREQ2}}$ should be configured.

Bits 18–23—Reserved

These bits are reserved and should be set to 0.

CBOE—Card B Output Enable

This bit is reflected on the OP2 pin that is used to three-state the external buffers when the card's power is activated. When the PCMCIA controller is in active mode:

0 = OP2 is low.
1 = OP2 is high.

CBRESET—Card B Reset

This bit is reflected on the OP3 pin used to reset the card. When the PCMCIA controller is in active mode:

0 = OP3 is low.
1 = OP3 is high.

Bits 26–31—Reserved

These bits are reserved and should be set to 0.

17.5.5 PCMCIA Base Registers

The PCMCIA base registers 0–7 (PBR0-7) contain the PCMCIA base addresses for the PCMCIA memory or I/O windows. The base registers are used in conjunction with the BSIZE field of the corresponding PCMCIA option register to ensure valid PCMCIA accesses.

PBR0–PBR7

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	PBA															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x80 (PBR0), 0x88 (PBR1), 0x90 (PBR2), 0x98 (PBR3), 0xA0 (PBR4), 0xA8 (PBR5), 0xB0 (PBR6), 0xB8 (PBR7)															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	PBA															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x80 (PBR0), 0x88 (PBR1), 0x90 (PBR2), 0x98 (PBR3), 0xA0 (PBR4), 0xA8 (PBR5), 0xB0 (PBR6), 0xB8 (PBR7)															

NOTE: — = Undefined.

PBA—PCMCIA Base Address

This field is compared to the address on the address bus to determine if a PCMCIA window is being accessed by an internal bus master. These bits are used in conjunction with the BSIZE field in the POR.

17.5.6 PCMCIA Option Registers

The PCMCIA option registers 0-7 (POR0-POR7) control the size, timing parameters, and memory access to the individual PCMCIA windows whose base addresses reside in the corresponding PCMCIA base registers.

POR0–POR7

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	BSIZE					RESERVED							PSHT			
RESET	—					—							—			
R/W	R/W					R/W							R/W			
ADDR	(IMMR & 0xFFFF0000) + 0x84 (POR0), 0x8C (POR1), 0x94 (POR2), 0x9C (POR3), 0xA4 (POR4), 0xAC (POR5), 0xB4 (POR6), 0xBC (POR7)															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	PSST				PSL					PPS	PRS			PSL OT	WP	PV
RESET	—				—					—	—			—	—	—
R/W	R/W				R/W					R/W	R/W			R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x84 (POR0), 0x8C (POR1), 0x94 (POR2), 0x9C (POR3), 0xA4 (POR4), 0xAC (POR5), 0xB4 (POR6), 0xBC (POR7)															

NOTE: — = Undefined.

BSIZE—PCMCIA Bank Size

This field determines the bank size (the size of the address space in bytes) for the corresponding PCMCIA window. The bank size is also used as an address mask and is applied to the PBA field in the associated PCMCIA base register to an address generated by an internal master. The bank size is calculated from BSIZE as:

$$\text{BankSize} = 2^{\text{GrayCode(BSIZE)}}$$

00000 = 1 bytes.
00001 = 2 bytes.
00011 = 4 bytes.
00010 = 8 bytes.

00110 = 16 bytes.
00111 = 32 bytes.
00101 = 64 bytes.
00100 = 128 bytes.

01100 = 256 bytes.
01101 = 512 bytes.

01111 = 1K.
01110 = 2K.

01010 = 4K.
01011 = 8K.
01001 = 16K.
01000 = 32K.

11000 = 64K.
11001 = 128K.
11011 = 256K.
11010 = 512K.

11110 = 1M.
11111 = 2M.
11101 = 4M.
11100 = 8M.

10100 = 16M.
10101 = 32M.
10111 = 64M.

The calculated bank size is used as a mask (MASK) to determine a valid PCMCIA address as follows:

```
If ((Address & MASK) == (PBA & MASK))
   Valid PCMCIA access
else
   invalid PCMCIA access
```

PSHT—PCMCIA Strobe Hold Time

This attribute is used to determine when IOWR_B or WE_B are negated during a PCMCIA write access or when IORD_B or OE_B are negated during a PCMCIA read access handled by the PCMCIA interface. This aids in meeting address/data hold time requirements for slow memories and peripherals.

0000= Strobe negation to Address change 0 clock.
0001= Strobe negation to Address change 1 clock.
0010= Strobe negation to Address change 2 clock.
0011= Strobe negation to Address change 3 clock.
0100= Strobe negation to Address change 4 clock.
0101= Strobe negation to Address change 5 clock.
0110= Strobe negation to Address change 6 clock.
0111= Strobe negation to Address change 7 clock.
1000= Strobe negation to Address change 8 clock.
1001= Strobe negation to Address change 9 clock.
1010= Strobe negation to Address change 10 clock.
1011= Strobe negation to Address change 11 clock.
1100= Strobe negation to Address change 12 clock.
1101= Strobe negation to Address change 13 clock.
1110= Strobe negation to Address change 14 clock.
1111= Strobe negation to Address change 15 clock.

PSST—PCMCIA Strobe Setup Time

This attribute is used to determine when IOWR_B or WE_B are asserted during a PCMCIA write access or when IORD_B or OE_B are asserted during a PCMCIA read access handled by the PCMCIA interface. This aids in meeting address/setup time requirements for slow memories and peripherals.

0000= Reserved
0001= Address to Strobe assertion 1 clock cycles.
0010= Address to Strobe assertion 2 clock cycles.
0011= Address to Strobe assertion 3 clock cycles.
0100= Address to Strobe assertion 4 clock cycles.
0101= Address to Strobe assertion 5 clock cycles.
0110= Address to Strobe assertion 6 clock cycles.
0111= Address to Strobe assertion 7 clock cycles.
1000= Address to Strobe assertion 8 clock cycles.
1001= Address to Strobe assertion 9 clock cycles.
1010= Address to Strobe assertion 10 clock cycles.
1011= Address to Strobe assertion 11 clock cycles.
1100= Address to Strobe assertion 12 clock cycles.
1101= Address to Strobe assertion 13 clock cycles.
1110= Address to Strobe assertion 14 clock cycles.
1111= Address to Strobe assertion 15 clock cycles.

PSL—PCMCIA Strobe Length

This attribute determines the number of cycles the strobe will be asserted during a PCMCIA access for this window. It is the main parameter for determining the length of the cycle. The cycle may be lengthened by asserting the $\overline{\text{WAIT}}$ signal.

00001= Strobe asserted 1 clock cycles.
00010= Strobe asserted 2 clock cycles.
00011= Strobe asserted 3 clock cycles.
00100= Strobe asserted 4 clock cycles.
00101= Strobe asserted 5 clock cycles.
00110= Strobe asserted 6 clock cycles.
00111= Strobe asserted 7 clock cycles.
01000= Strobe asserted 8 clock cycles.
01001= Strobe asserted 9 clock cycles.
01010= Strobe asserted 10 clock cycles.
01011= Strobe asserted 11 clock cycles.
01100= Strobe asserted 12 clock cycles.
01101= Strobe asserted 13 clock cycles.
01110= Strobe asserted 14 clock cycles.
01111= Strobe asserted 15 clock cycles.
10000= Strobe asserted 16 clock cycles.
10001= Strobe asserted 1 7 clock cycles.
10010= Strobe asserted 18 clock cycles.
10011= Strobe asserted 19 clock cycles.
10100= Strobe asserted 20 clock cycles.
10101= Strobe asserted 21 clock cycles.
10110= Strobe asserted 22 clock cycles.
10111= Strobe asserted 23 clock cycles.
11000= Strobe asserted 24 clock cycles.
11001= Strobe asserted 25 clock cycles.
11010= Strobe asserted 26 clock cycles.
11011= Strobe asserted 27 clock cycles.
11100= Strobe asserted 28 clock cycles.
11101= Strobe asserted 29 clock cycles.
11110= Strobe asserted 30 clock cycles.
11111= Strobe asserted 31 clock cycles.
00000= Strobe asserted 32 clock cycles.

PPS—PCMCIA Port Size

This field specifies the port size of this PCMCIA window.

0 = 8-bit port size.
1 = 16-bit port size.

PRS—PCMCIA Region Select

000 = Common memory space.
001 = Reserved.
010 = Attribute memory space.
011 = I /O space.
100 = DMA (normal DMA transfer.
101 = DMA last transaction.
110 = Drive the value of the A22 and A23 signals on CE2 and CE1.
111 = Reserved.

The "DMA" encoding will generate a normal DMA transfer unless signaled as "last" by the on-chip DMA controller. In this case, TC($\overline{OE}$) or TC ($\overline{WE}$) is asserted. The "DMA last transaction" encoding will generate a DMA transfer with TC($\overline{OE}$) or TC ($\overline{WE}$) asserted, regardless of any internal indication.

PSLOT—PCMCIA Slot Identifier

0 = Reserved.
1 = This window defined for slot B.

WP—Write-Protect

0 = This window is not write-protected.
1 = This window is write-protected. If you try to write to this window, a bus error (machine check interrupt) will occur.

PV—PCMCIA Valid

This bit indicates that the contents of the PCMCIA base register and option register pair are valid.

0 = This bank is invalid.
1 = This bank is valid

17.6 PCMCIA CONTROLLER TIMING EXAMPLES

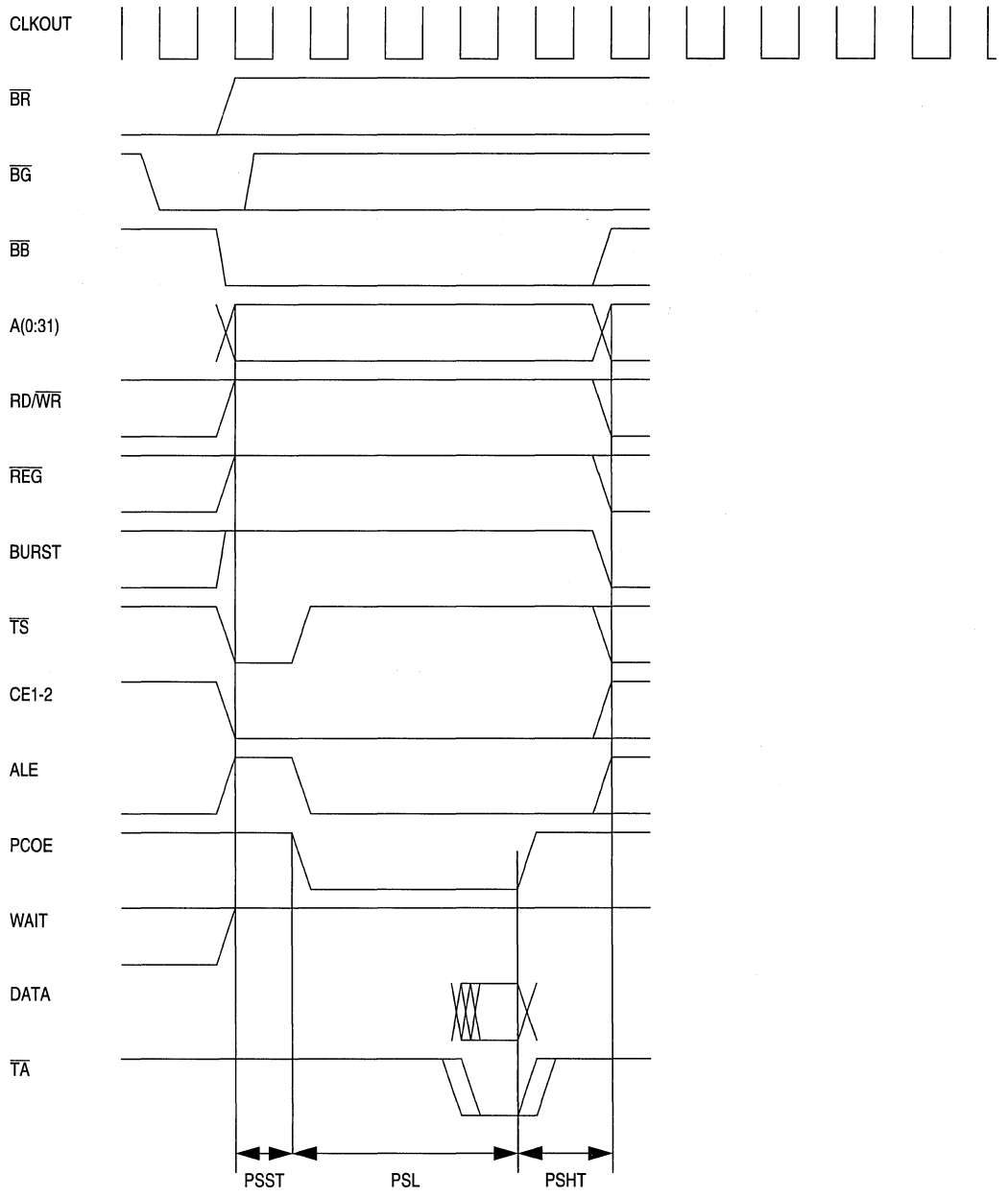


Figure 17-3. PCMCIA Single Beat Read Cycle PRS = 0 PSST = 1 PSL = 3 PSHT = 1

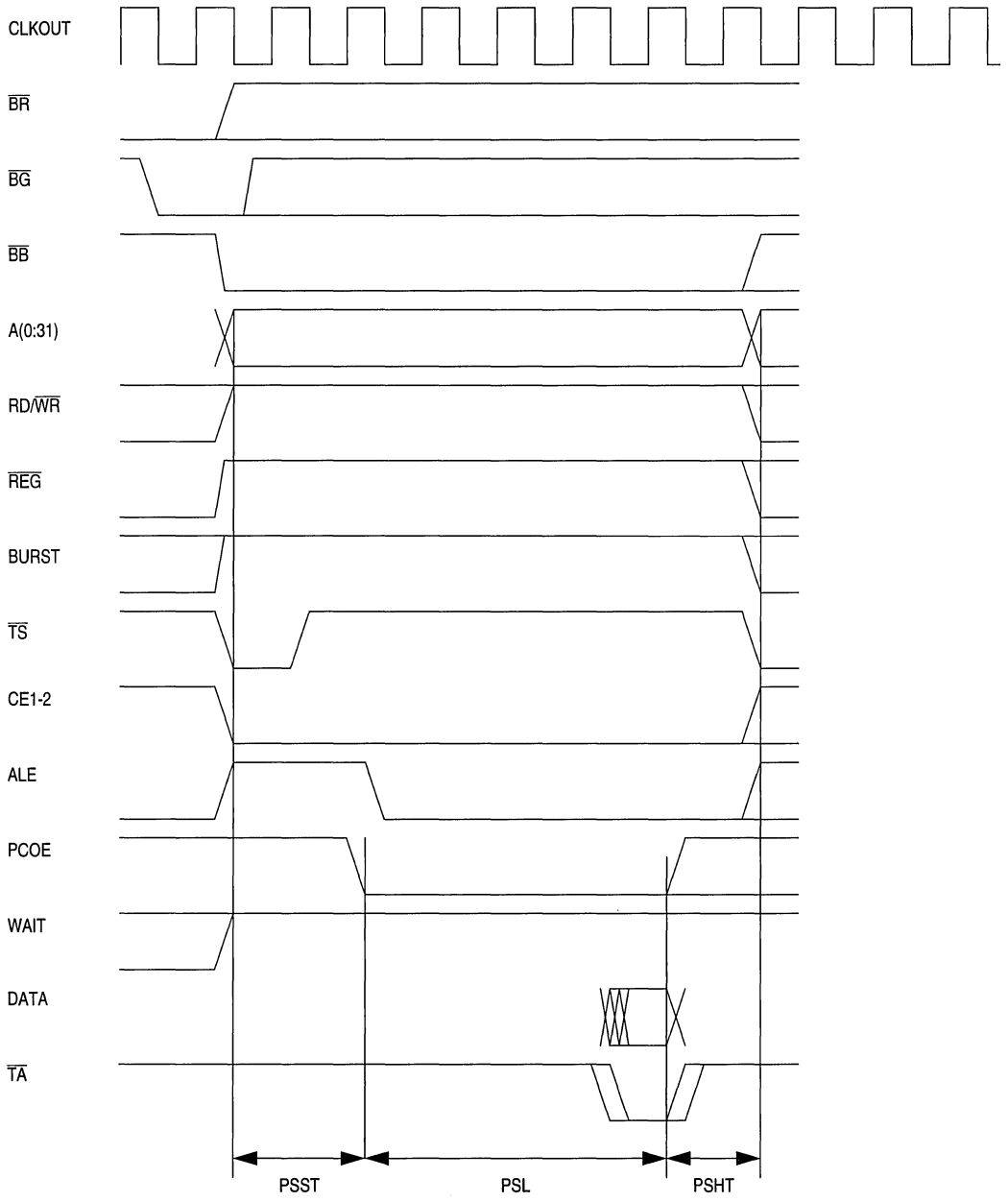


Figure 17-4. PCMCIA Single Beat Read Cycle PRS = 0 PSST = 2 PSL = 4 PSHT = 1

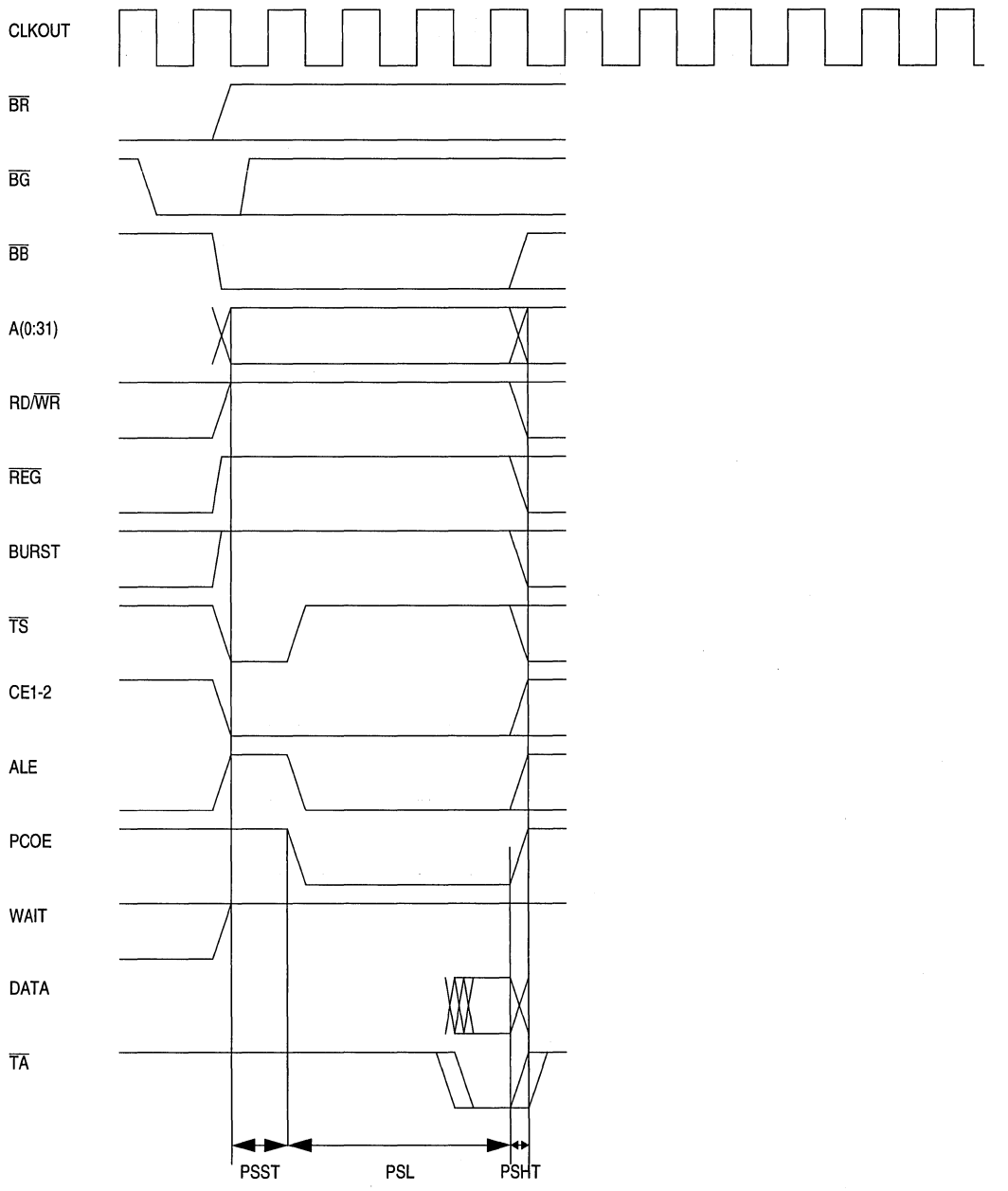


Figure 17-5. PCMCIA Single Beat Read Cycle PRS = 0 PSST = 1 PSL = 3 PSHT = 0

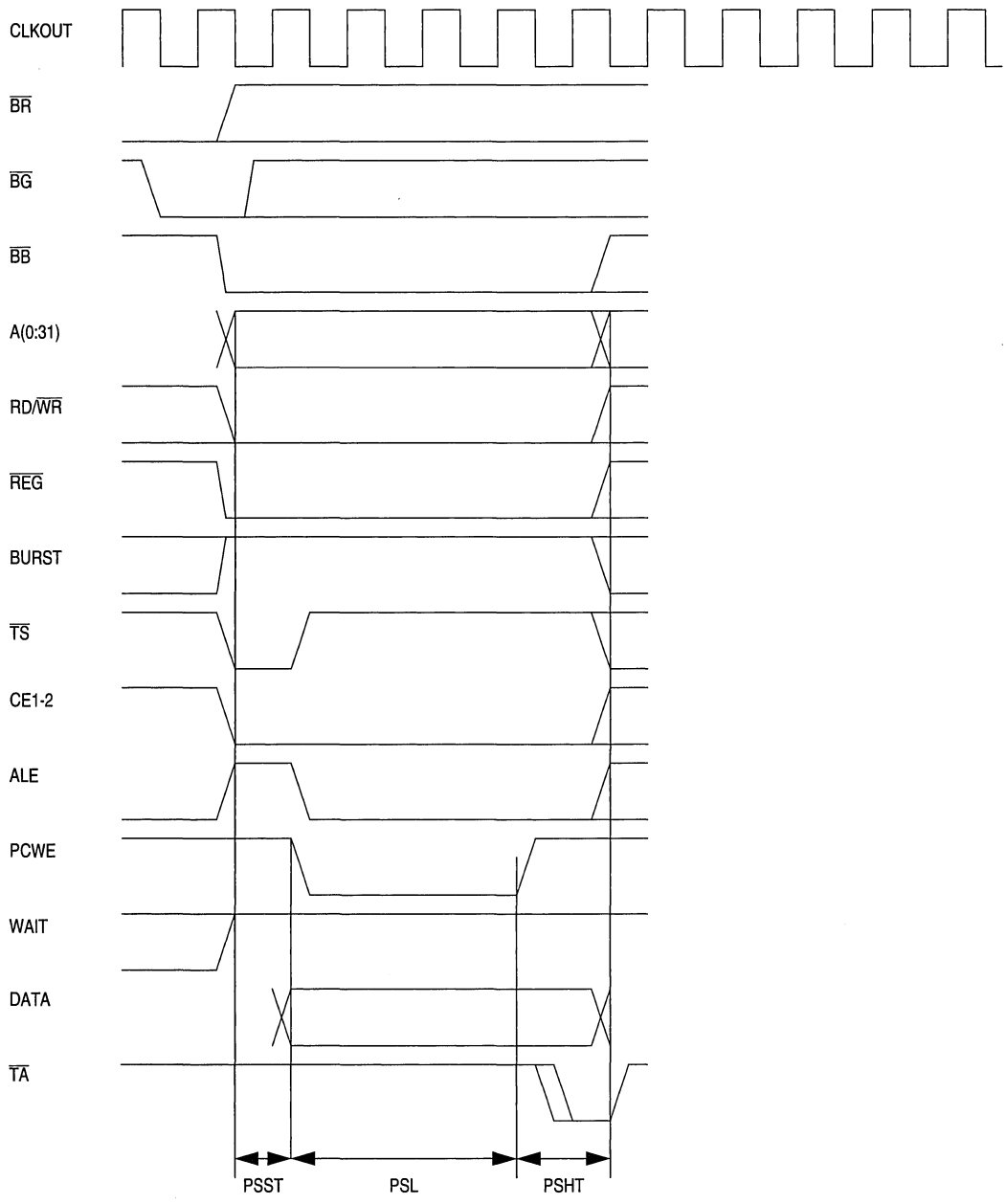


Figure 17-6. PCMCIA Single Beat Write Cycle PRS = 2 PSST = 1 PSL = 3 PSHT = 1

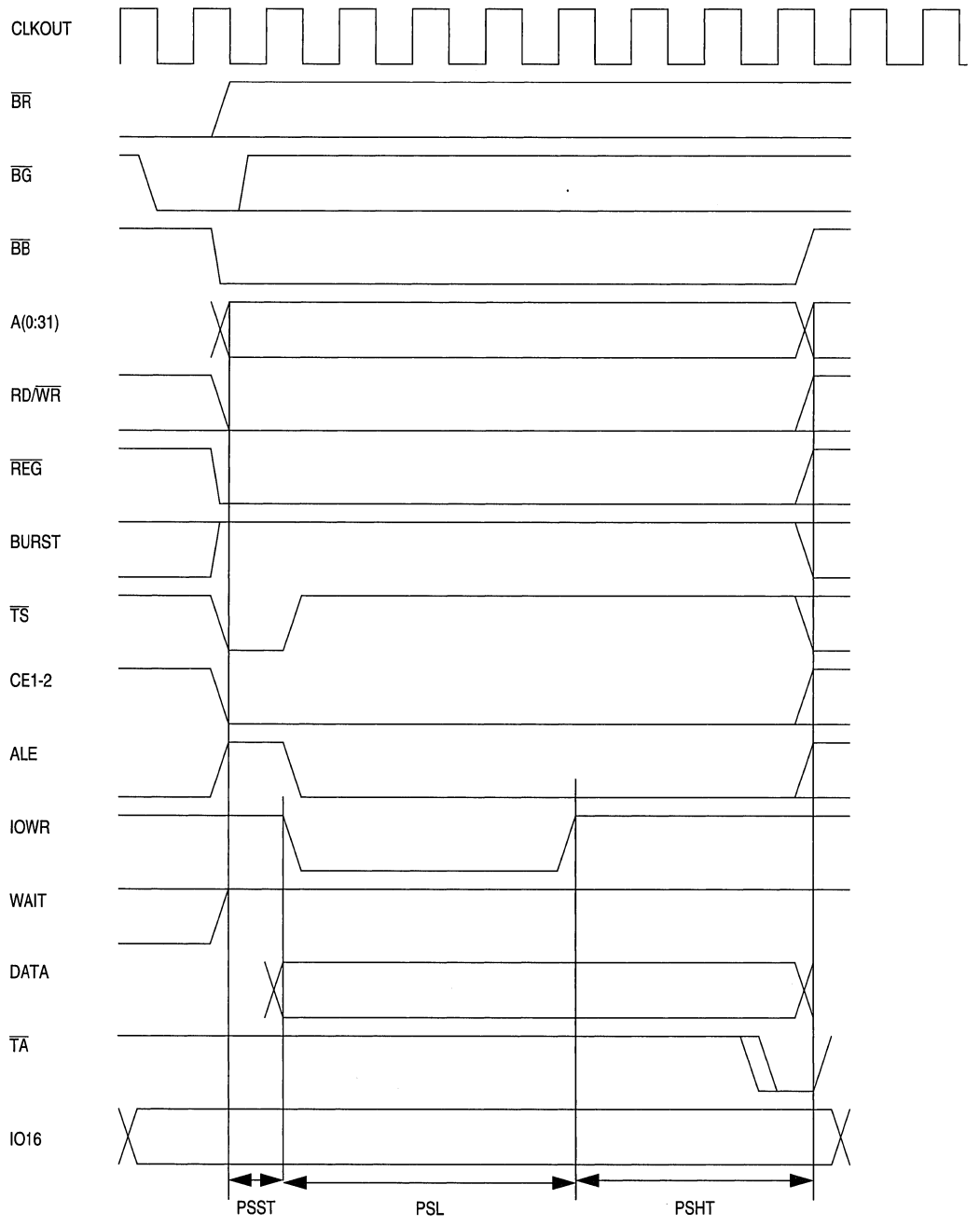


Figure 17-7. PCMCIA Single Beat Write Cycle PRS = 3 PSST = 1 PSL = 4 PSHT = 3

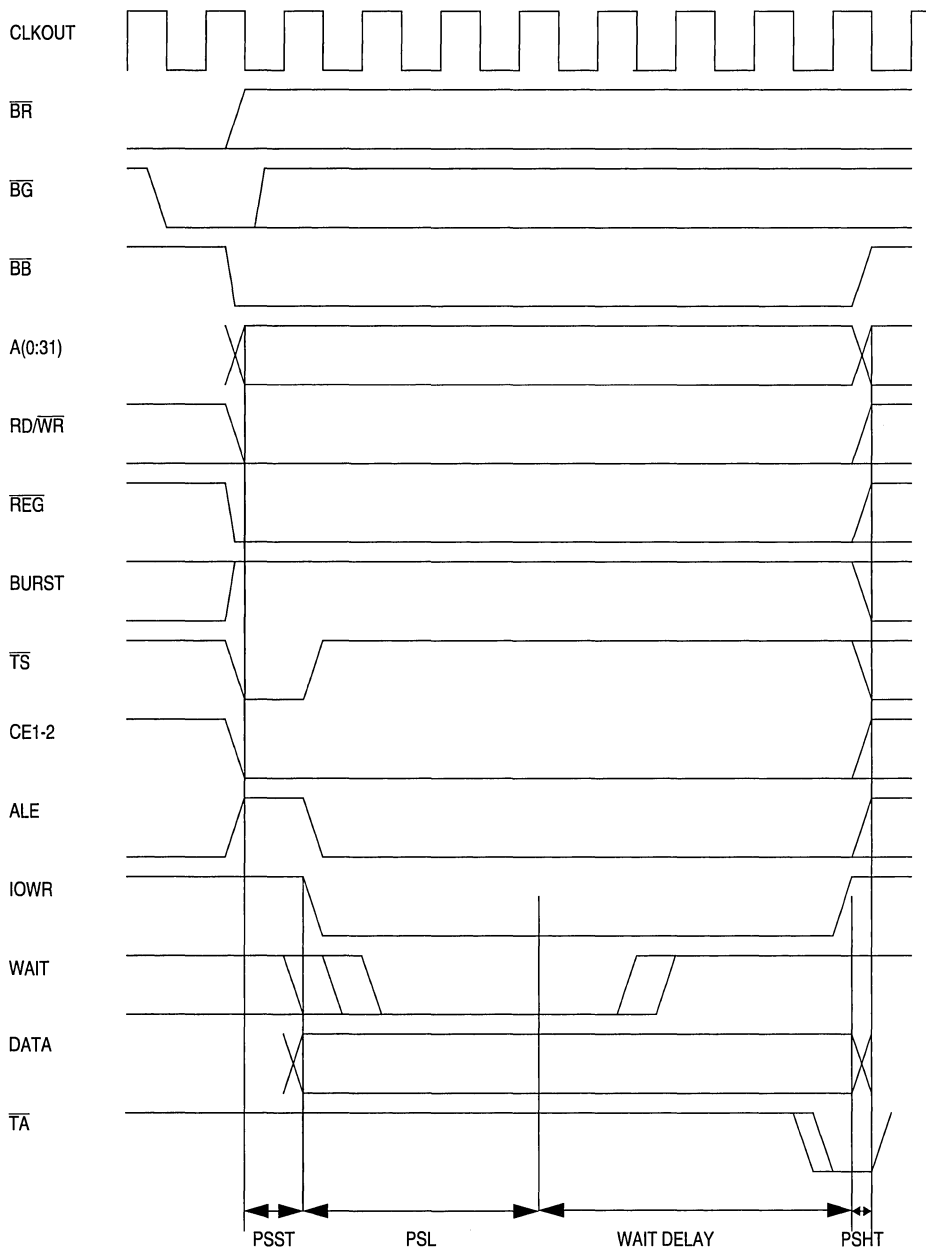


Figure 17-8. PCMCIA Single Beat Write with Wait PRS = 3 PSST = 1 PSL = 3 PSHT = 0

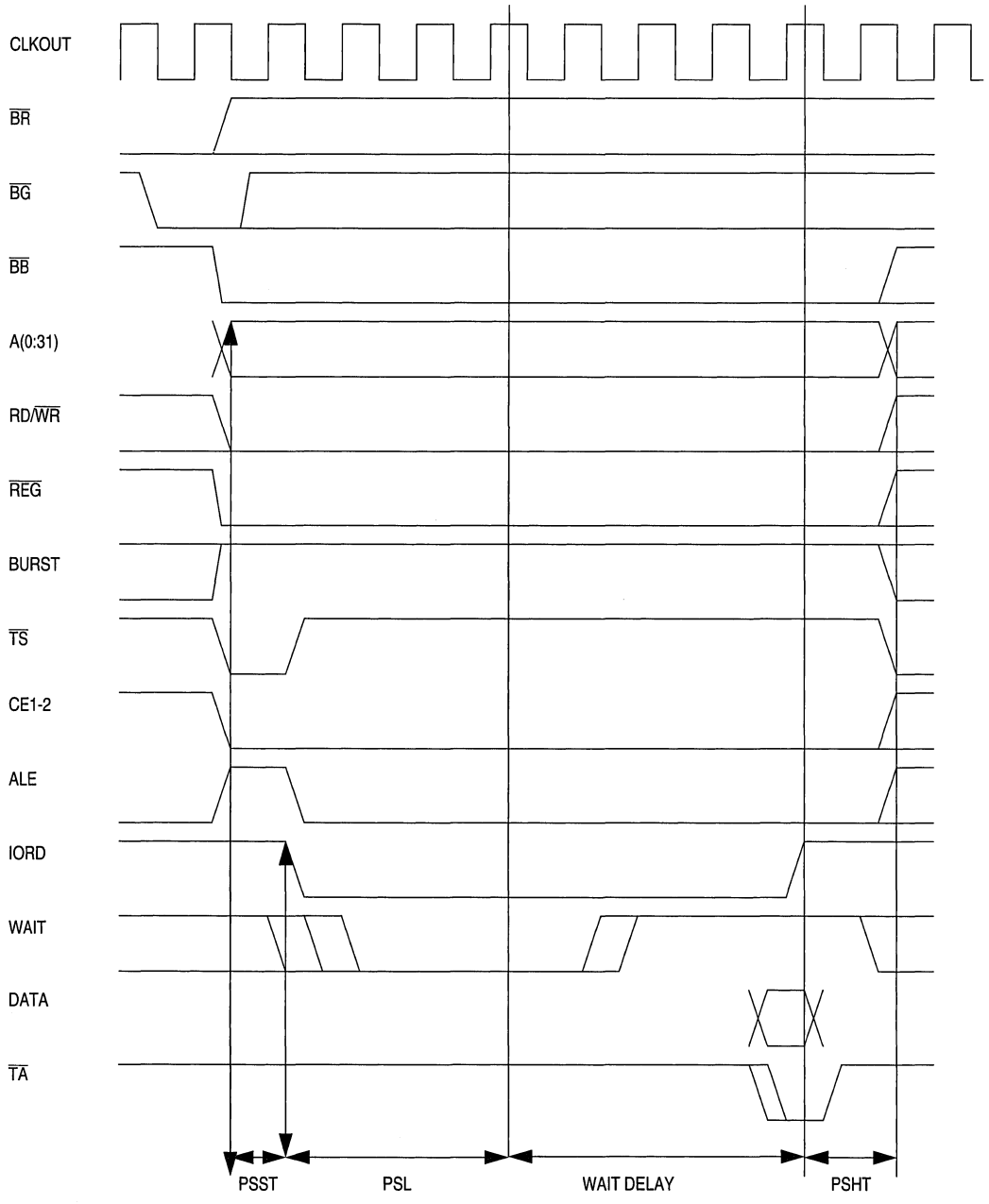


Figure 17-9. PCMCIA Single Beat Read with Wait PRS = 3 PSST = 1 PSL = 3 PSHT = 1

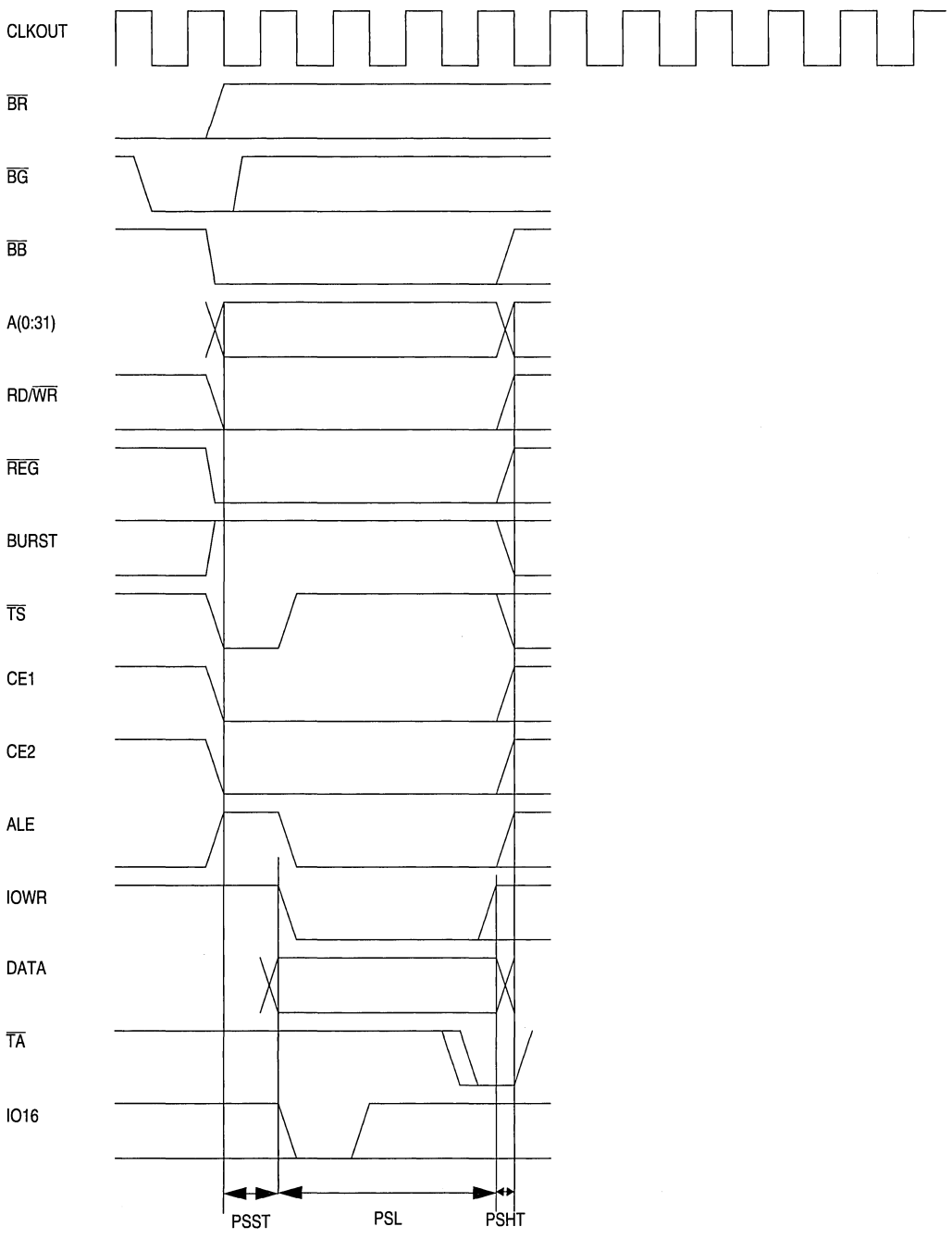


Figure 17-10. PCMCIA I/O Read PPS = 1PRS = 3 PSST = 1PSL = 2 PSHT = 0

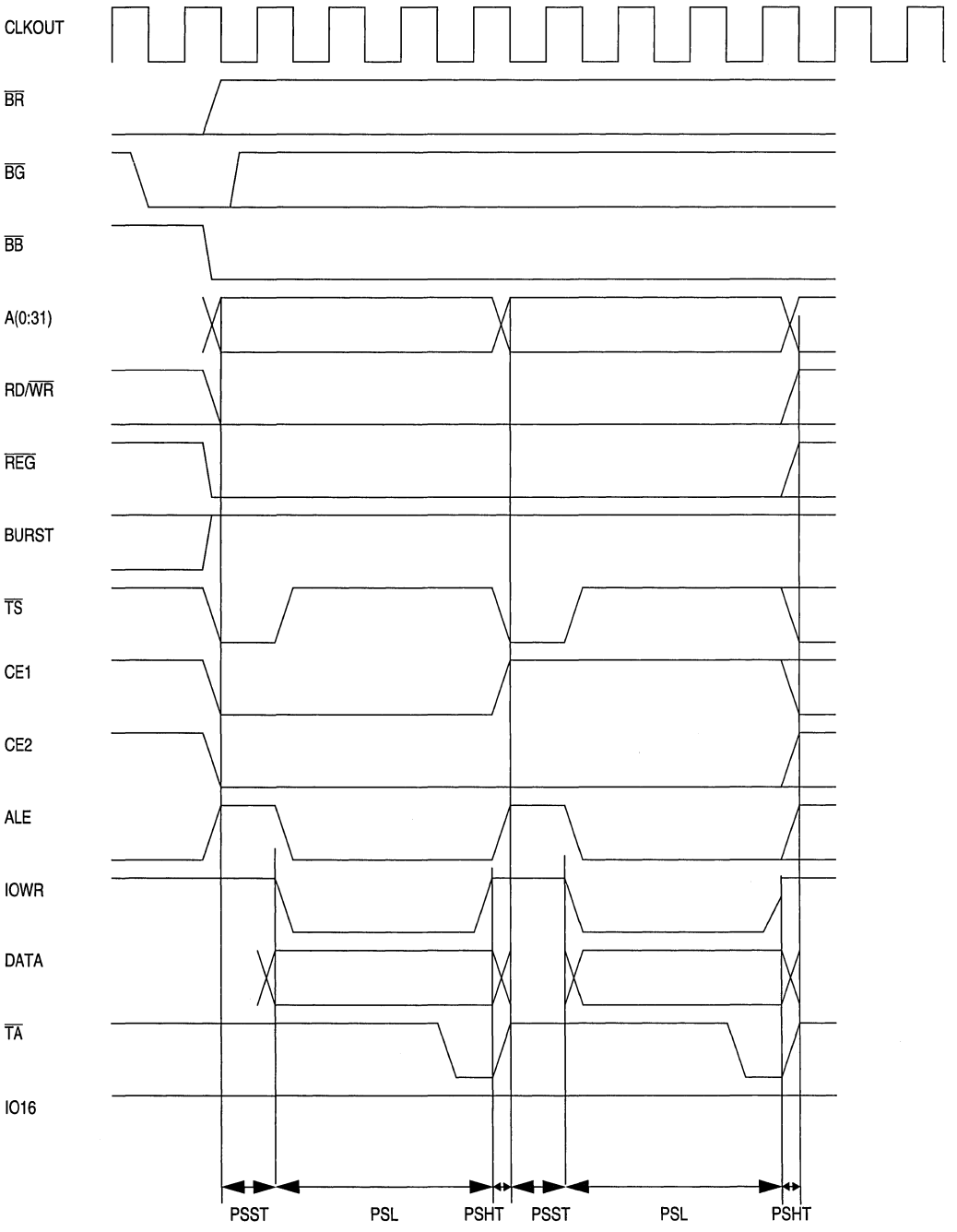


Figure 17-11. PCMCIA I/O Read PPS = 1 PRS = 3 PSST = 1 PSL = 2 PSHT = 0

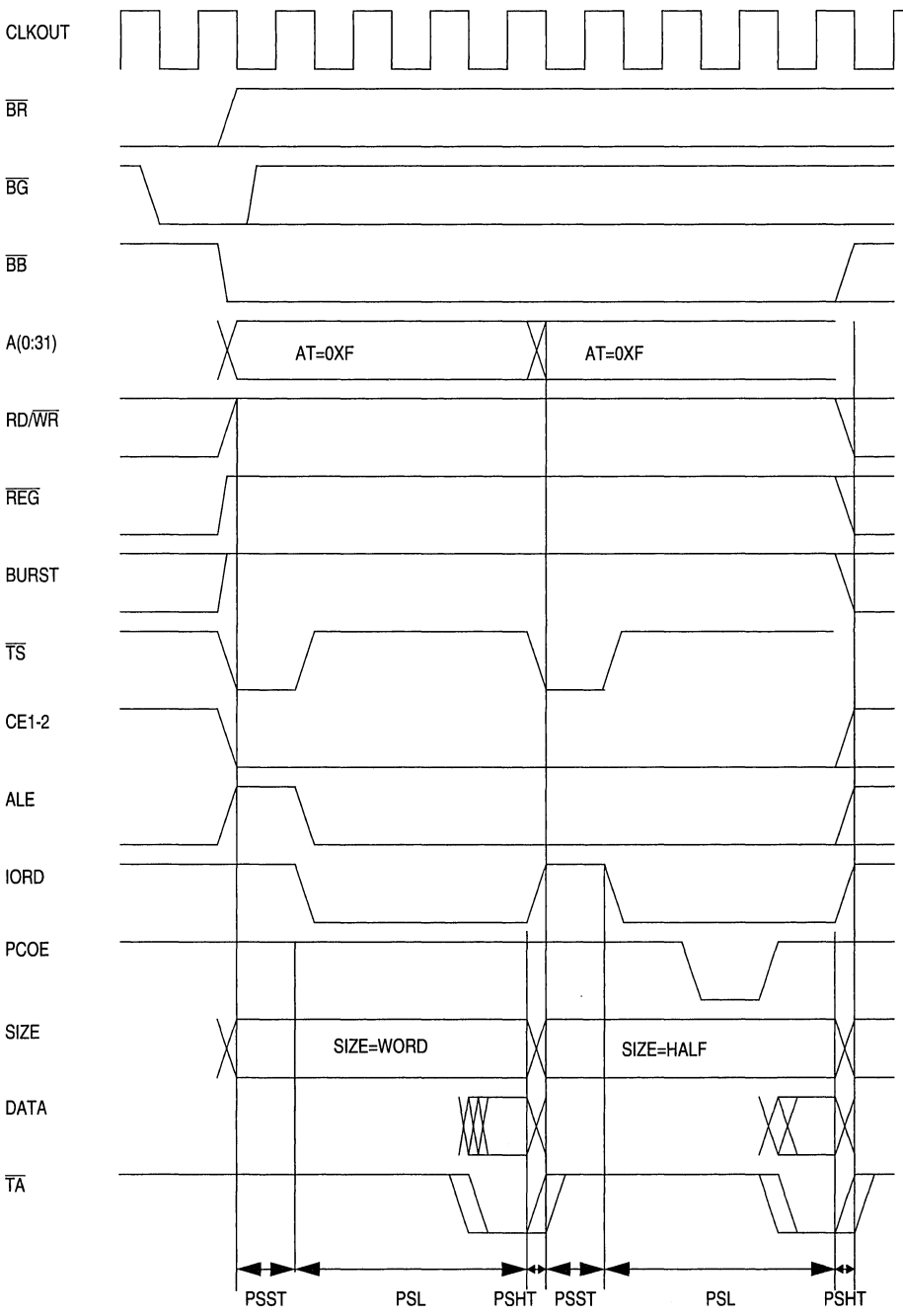


Figure 17-12. PCMCIA DMA Read Cycle PRS = 4 PSST = 1 PSL = 3 PSHT = 0

SECTION 18
LCD CONTROLLER

The MPC823 contains an on-chip LCD controller that can be used to drive an LCD panel display. The integrated LCD controller shortens access time, reduces power consumption, and saves system board space by not using external glue logic. The LCD controller can interface with a variety of passive, active, dual-scan, single-scan, and smart LCD panels.

18.1 FEATURES

The following is a list of the LCD controller's main features:

- Passive, Active, and Smart Panel Interfaces Are Supported
- Requires No Special Display Memory (Uses System Memory)
- One, Two, or Four Bits Per Pixel in Grayscale Mode Generates a Maximum of 16 Grayscale Levels Using an Advanced Frame Rate Control Algorithm
- Four or Eight Bits Per Pixel in Color Mode Generates 16 or 256 Simultaneous Colors From 4,096 Color Patterns
- Interfaces To the LCD Panel Through a 4-, 8-, or 12-Bit Wide Parallel Output Bus
- Supports Single-Scan or Dual-Scan Screens
- Supports Many Split Display Data Modes
- Requires No CPU or Communication Processor Module Intervention
- Built-In 256 Entry Color RAM
- Programmable Wait Time Between Lines and Frames
- Programmable Panel Voltage Control for Brightness and Contrast
- Programmable Polarity For All LCD Interface Signals
- TFT/RGB Output Drives Advanced Buffer LCD Driver Chips
- Bus Performance Optimized with Burst Read DMA Cycles

18.1.1 LCD Technology

A liquid crystal display (LCD) implements a low-power display technology that uses ambient light to display images. LCDs consist of two pieces of glass with electrodes printed on the inside and polarizers are used on the outside front and rear surfaces. When the LCD is off, no voltage is applied to the electrodes and light passes through the LCD. When it is on, voltage is applied to the electrodes and the liquid crystal molecules align themselves in the direction of the electric field. This causes the light to be blocked and out of phase with the polarizers, which creates a dark area on the LCD. This darkness is a function of the RMS voltage applied to them. This means that the polarity of the applied voltage is not important and that no DC bias is allowed. A typical monochrome LCD module is illustrated in Figure 18-1 below.

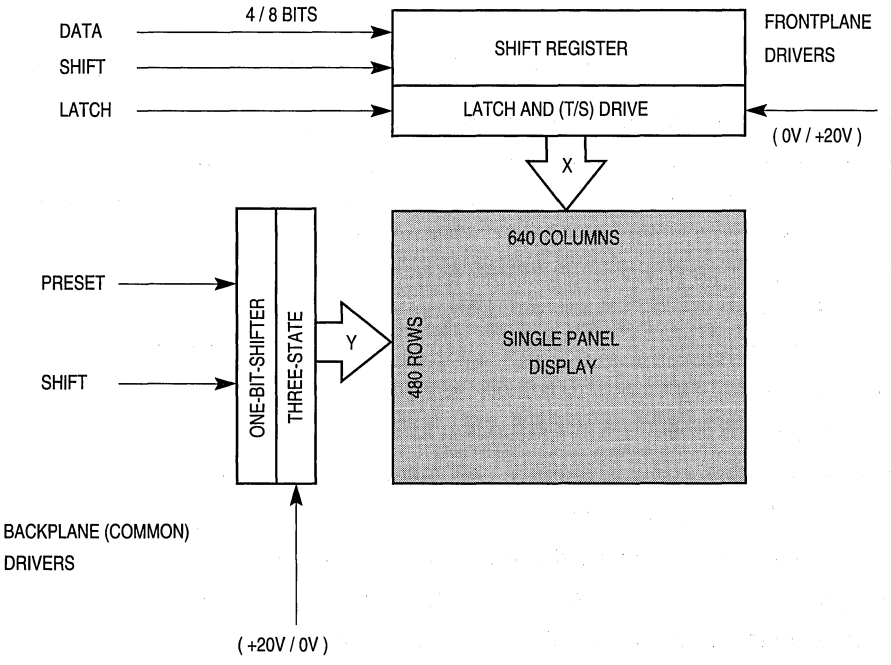


Figure 18-1. LCD Panel

The Y drivers apply high voltage to one row at a time. At the same time, the X drivers connect the dark pixels to the opposing voltage and bright pixels to the same voltage as the row. The value of each pixel is shifted by the shift register and latched when the whole row is ready. While one row is being refreshed, the next row is being shifted in and each row is refreshed at a rate inverse to the number of rows. If the number of rows is large, then the refresh rate might become too small. When this occurs, the panel is divided into two sections (upper and lower). Then the X shift register, which is twice as long, works at twice the speed to refresh two columns at the same time (one column in the upper section and one in the lower section). The X shift register can be loaded 1, 2, 4, or 8 bits per shift.

The time it takes to refresh a row is called the duty cycle. In a simple refresh policy, one row is refreshed at a time. However, this creates large voltage spikes on the LCD electrodes. Another technique, called active addressing, refreshes groups of rows and smoothes the applied voltage that each pixel receives over approximately half a frame cycle. This significantly improves the contrast and other characteristics of a simple LCD display. The drawback of active addressing is that it is fairly complex and requires that you have a dedicated chip to perform it on. This chip holds a large display buffer (typically equivalent to a quarter of a VGA display per circuit). The active addressing algorithm is done on entire columns and it usually fits in the LCD display path and appears to the system as a thin film transistor (TFT) interface.

It is important to avoid long-term DC bias in the voltages applied to the LCD screen because it causes defects in the polarization as well as raindrop-like patterns (raindrop effect) to appear on the display. To overcome this problem the controller should change the field polarity every few frames.

18.1.2 Types of LCD Interfaces

Basic LCD panels require a clock, one or more data input lines, and horizontal and vertical syncs. These signals are usually provided by an LCD controller that includes a frame buffer RAM for display memory. An example of a complete LCD subsystem is illustrated in Figure 18-2.

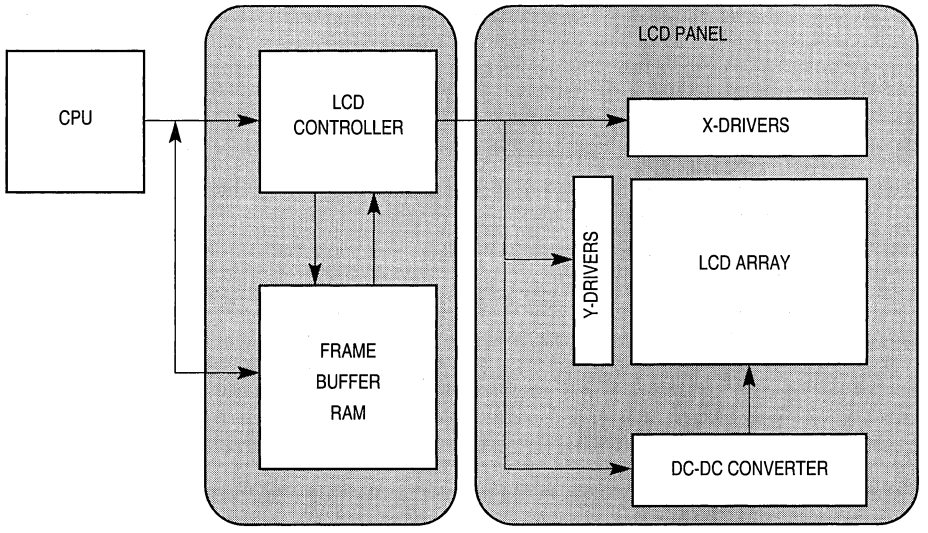


Figure 18-2. LCD Subsystem

18.1.2.1 PASSIVE LCD INTERFACE. A passive LCD panel interface uses X and Y shift registers to operate. The X shift register is used to display a column and the Y shift register is used to display a row. The LCD controller fills the shift register, provides framing, and reverses the display polarity from frame-to-frame or line-to-line.

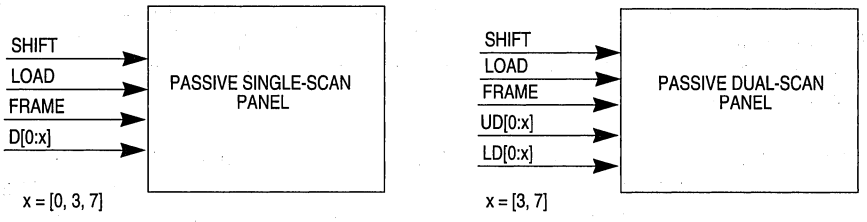


Figure 18-3. Passive Interfaces

A passive LCD interface consists of several parallel data bits that are shifted into the X shift register by the shift clock. After the shift register is full, a latch signal transfers the pixels from the shift register to driver latches and moves the Y pointer down one line. At this point, the shift operation continues to the next line and after all the lines are scanned, the frame signal moves the Y pointer to the beginning of the frame. The LCD controller also accesses the frame buffer. The panel can be single- or dual-scan and the dual-scan function is accomplished by splitting the Y dimension. For single-scan panels, the LCD controller only has one buffer. For dual-scan panels, the LCD controller must have one upper and one lower frame buffer. In most LCD panels, you need control to kill any DC biases that build up during normal operation. A signal that inverts the polarity of the voltages is presented to the LCD panel. Usually, this signal toggles every few frames (1–20).

18.1.2.2 ACTIVE LCD INTERFACE. An active LCD panel interface is referred to as a thin film transistor (TFT) interface. It provides a high-performance LCD panel that looks more like a digital RGB or monochrome video signal that has several data bits in parallel. The shift clock is also present. Latch and frame signals are called horizontal sync and vertical sync and have special timing. There is also a special signal called output enable that blanks/enables data, but does not affect the clock. For color displays, the MPC823 supports a 12-bit (four bits per basic color) data bus. For example, an 8-bit pixel data fetched from the frame buffer is passed through a 256 x 12 memory that selects one out of 256 colors.

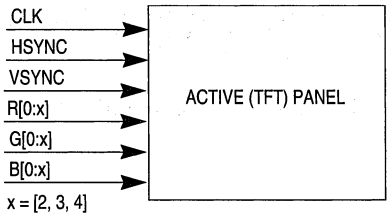


Figure 18-4. Active (TFT) Interface

18.1.2.3 SMART PANEL LCD INTERFACE. In a smart panel interface, the whole memory display buffer resides on the panel, which is directly connected to the system bus. Also, the CPU accesses the display memory directly, so you do not need a controller.

18.2 THE MPC823 LCD CONTROLLER

The MPC823 LCD controller is initialized by the core, which provides the frame buffer address, operational modes, and various configuration bits that the LCD controller needs to operate. After it is enabled, the LCD controller requests the DMA to fetch the frame buffer data. The frame buffer is always organized in rows and columns. Depending on the interface you are using, the data is then interpreted for grayness or color and frame format. The data is then packed according to the model you have chosen. If you are using a split panel display, you need to initialize two buffers—one for each panel display. The LCD controller uses continuous DMA to feed the display. Figure 18-5 illustrates a typical MPC823 LCD system.

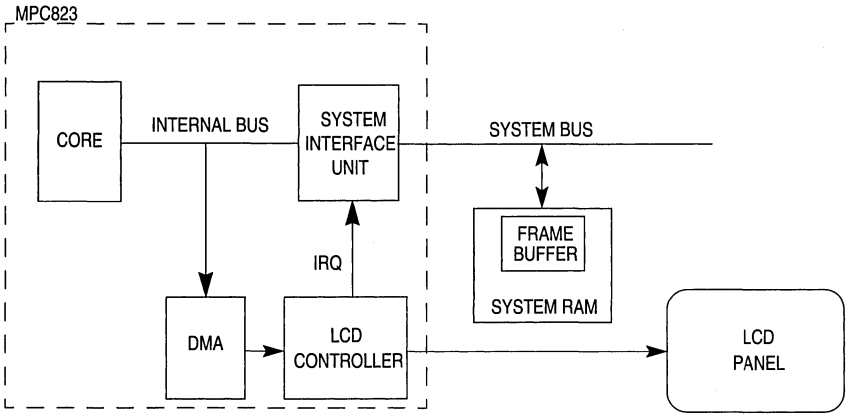


Figure 18-5. The MPC823 LCD System

The MPC823 LCD controller is a module that is separate from the core and communication processor module. It uses the internal DMA, bus, and system interface unit to access frame buffer memory. The LCD controller has dedicated registers that provide timing generation to control the panel. It also has two FIFOs that interface to color RAM, which provides pixel generation and a data path to the panel. The LCD controller block diagram is illustrated in Figure 18-6.

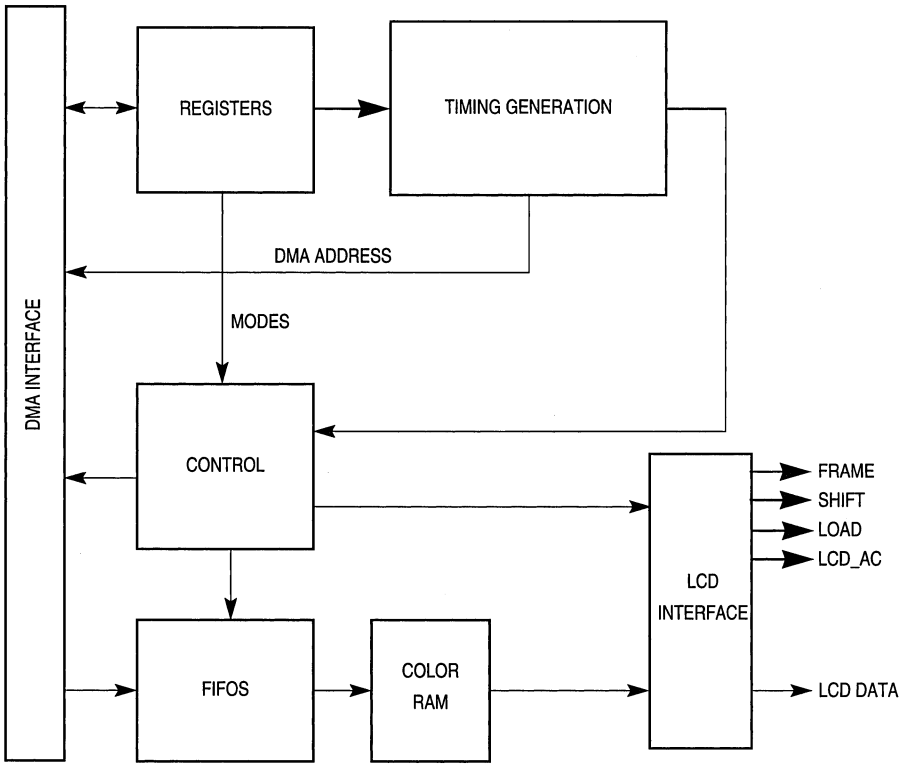


Figure 18-6. LCD Controller Block Diagram

18.3 LCD CONTROLLER OPERATION

The LCD controller uses the system interface unit to communicate with the core and external system. It has its own DMA functionality to fetch display memory into the FIFOs for pixel generation. The LCDCLK signal is derived from the SPLL output (VCOOUT) and is fed to the timing generator for vertical, horizontal, and frame timing. The register set is used to program the timing parameters for an LCD panel. Figure 18-7 illustrates the various modules of the LCD controller.

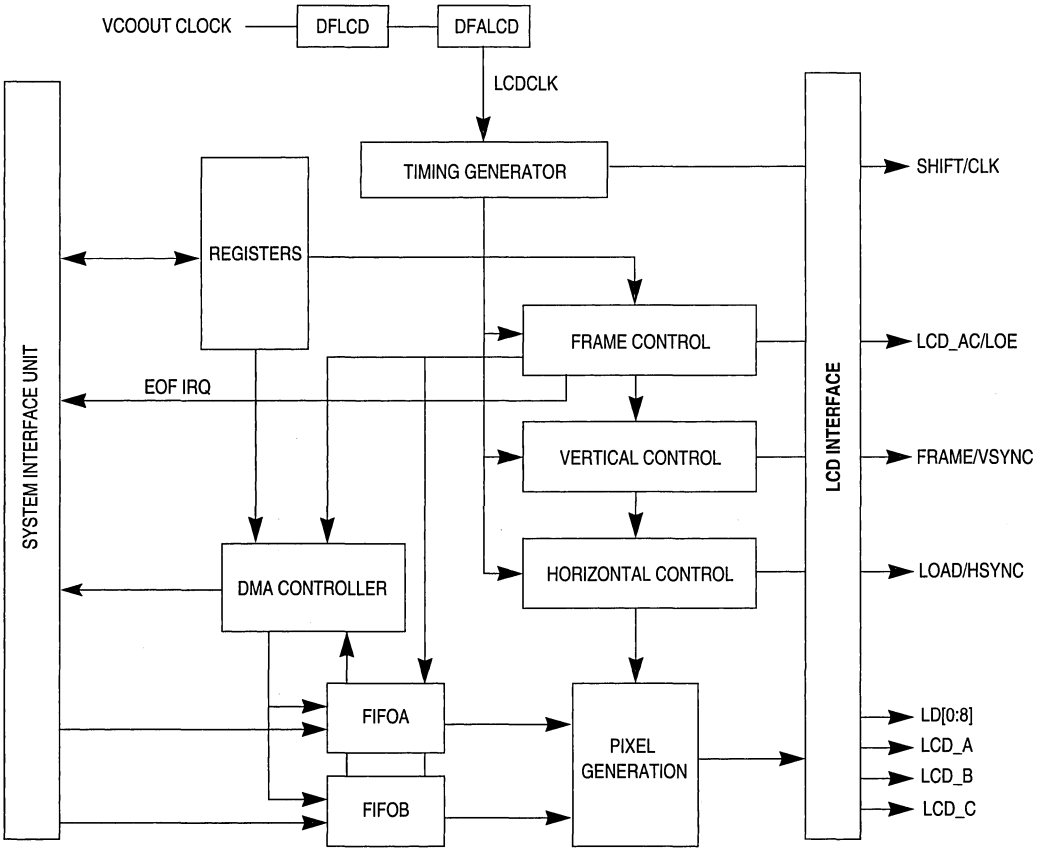


Figure 18-7. LCD Functional Module

After reset, the LCD controller is disabled (PON=0) in the LCD configuration and control register (LCCR). To operate the LCD controller, you should program the LCD registers and set the PON bit. When enabled, the FIFOs ask the DMA controller to fill them using burst read memory cycles. When data is available in the FIFOs, it is used to index into the color RAM to produce a grayscale or color pattern. This pattern is then shifted out by the horizontal control block, which generates timing for each pixel on a line, including the wait between lines (WBL). The vertical control block counts the number of lines and provides a wait between frames (WBF) timing.

The LCD controller consists of eight main blocks:

- FIFOs
- Pixel generation
- Horizontal control
- Vertical control
- Frame control
- DMA control
- Timing control
- LCD interface

18.3.1 FIFO Control

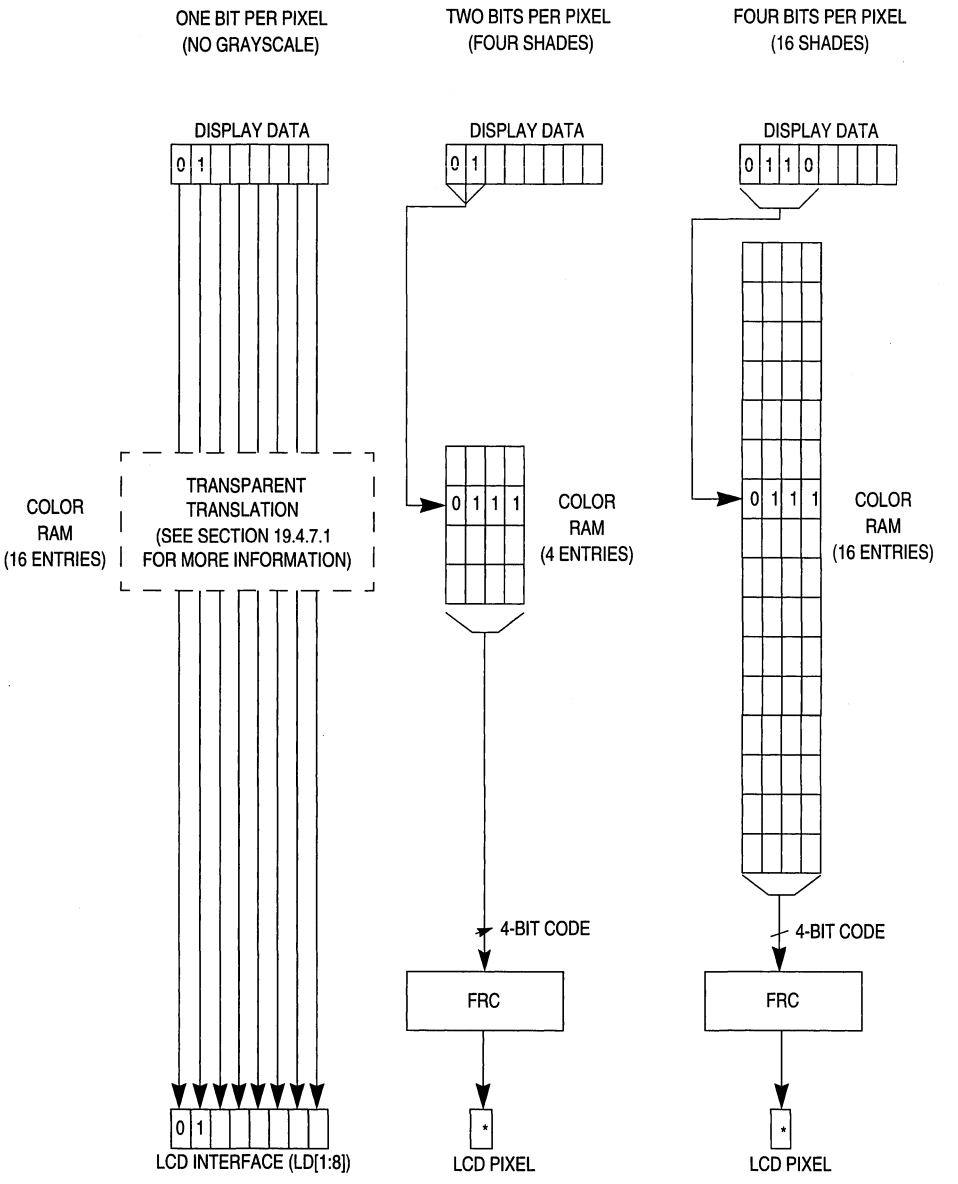
There are two FIFOs in the LCD controller that are concatenated for single-scan displays and used separately for dual-scan displays. Each FIFO has the capacity to hold 12 32-bit words. When the FIFO can accept a burst, it requests a DMA controller access, which guarantees no FIFO overflow. When the FIFO is empty before frame completion and the DMA cannot provide data because of heavy bus loading, a FIFO underrun condition occurs. If such a condition occurs, the display image may flicker, tear, or shift. In order to recover, you must restart the LCD controller. It is your responsibility to make sure the LCD controller has sufficient bus bandwidth to avoid underrun conditions. When the data is available in the FIFOs, the frame controller initiates frame processing.

18.3.2 Pixel Generation

The pixel generation block retrieves data from the FIFOs to generate a grayscale or color pattern by indexing into the color RAM area of dual-port RAM. For single-scan displays, data is fetched serially from both FIFOs. For dual-scan displays, data is fetched alternatively from both FIFOs.

18.3.2.1 GRAYSCALE. For single-bit grayscale (monochrome), each pixel is represented by a single bit in the frame buffer, also called the display buffer. A zero signifies pixel off and a one signifies pixel on. For fours bits per pixel, one pixel requires four bits in the memory that are packed together with the next four bits for the next pixel. The LCD controller can generate a maximum of 16 grayscale levels whereby each pixel is represented by four bits in the frame buffer. Grayscale is generated by controlling the rate in which each pixel is turned off and on. This method is referred to as frame rate control (FRC). Turning the pixels on and off under frame rate control may result in a low frequency pixel refresh rate to a given pixel. This refresh rate may cause image flickering when the same gray level is found in adjacent pixels. However, the MPC823-implemented FRC algorithm should minimize this flicker.

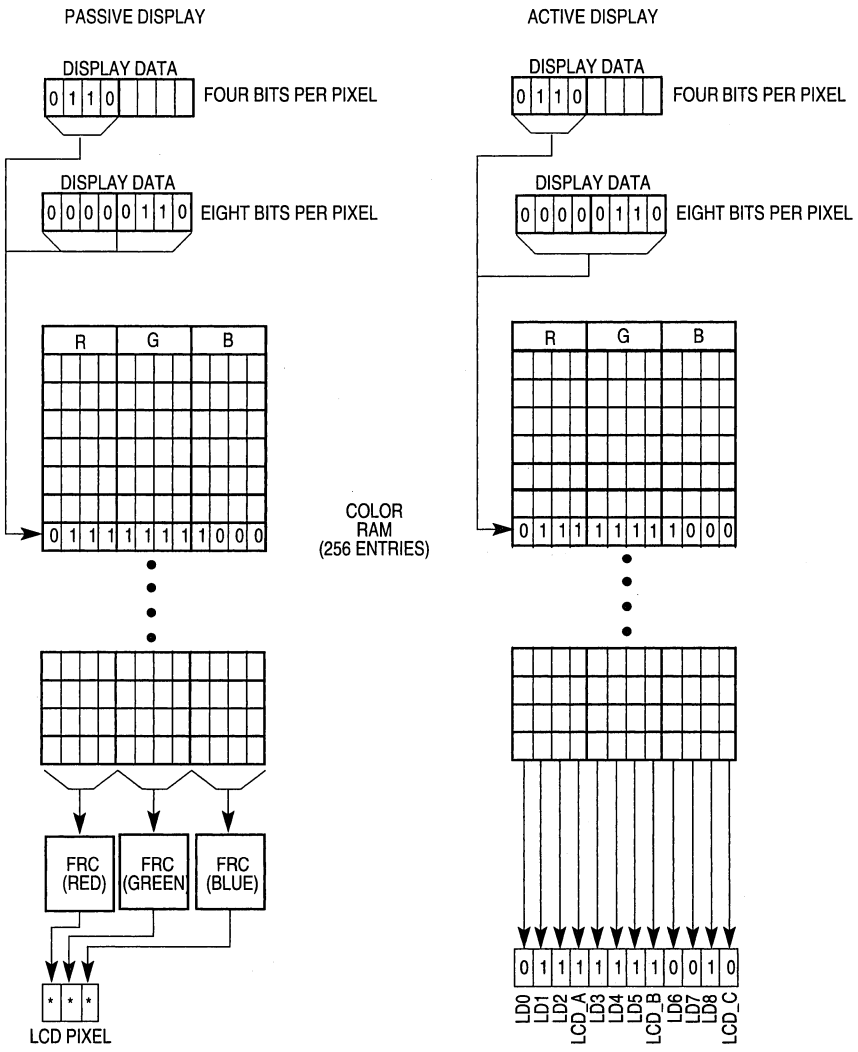
When one bit per pixel is defined, the color RAM table is used as a transparent translation to pass the RAM data through the LCD panel interface. When you are using two bits per pixel, the 2-bit code is indexed into the color RAM to one of four entries. Each of these four entries can be one of 16 possible gray level codes. These four entries are then used to determine the FRC value for that pixel. With four bits per pixel, the 4- bit code is used to index one of 16 entries in the color RAM. The 4-bit code from the color RAM is then used to determine one of the 16 shades generated by frame rate control. Figure 18-8 illustrates grayscale generation.



NOTE: * FOR EACH FRAME IN A GROUP OF 16, 1 INDICATES THIS PIXEL IS ON AND 0 INDICATES THIS PIXEL IS OFF.

Figure 18-8. Grayscale Generation

18.3.2.2 COLOR. Each color pixel is represented as a 4- or 8-bit code in the frame buffer. Using the color RAM, the pixel code is mapped to a 12-bit red/green/blue (RGB) code, which allows you to select one of 4,096 colors. For passive color displays, the frame rate control algorithm processes each color to generate the required amount of intensity, as derived from the nine bits (maximum) of RGB information in the color RAM. For active color displays, 12 bits (4 bits per color) are output directly onto the LCD data bus. Figure 18-9 illustrates color generation. Notice that the LCD data bus is shown with MSB on the left and LSB on the right. The LCD_A, LCD_B, and LCD_C bits are the least-significant bits of each color.,



NOTE: * FOR EACH FRAME IN A GROUP OF 16, 1 INDICATES THIS COLOR PIXEL IS ON AND 0 INDICATES THIS COLOR PIXEL IS OFF.

Figure 18-9. Color Generation

18.3.3 Horizontal Control

The horizontal control block tracks the pixel count for one line and any additional wait between lines. It also enables the pixel generation block by passing a signal from the vertical control block to each line. Upon completion of the wait between lines, as indicated in the WBL field of the LCHCR, it signals the vertical control block and activates the LOAD/HSYNC signal to indicate the start of the next line.

18.3.4 Vertical Control

The vertical control block counts the lines and signals the horizontal control block. Upon completion of the wait between frames, as indicated in the WBF field in the LCVCR, it signals the frame control block and activates the FRAME/VSYNC signal to indicate the start of the next frame.

18.3.5 Frame Control

The frame control block initializes all counters, starts the DMA, and provides signaling to the vertical control block. The frame control block generates an end of frame signal indicator that can be used to generate an interrupt. The end of frame interrupt can be enabled by setting the IEN bit in the LCCR.

18.3.6 DMA Control

The DMA control block handles all data transfers to and from the FIFOs and keeps them filled. At the appropriate time, each FIFO is filled by the DMA control block in 16-byte transfers. Your frame buffer address should be 16-byte aligned. When the LCD controller is initiated at reset or after an underrun condition, LCD data transfers to the panel start after five DMA read burst accesses to display memory have filled the FIFOs. See **Section 16.5.1 SDMA Bus Arbitration and Transfers** for proper arbitration configuration between the LCD controller DMA and the SDMA.

18.3.7 Timing Control

The LCD controller is clocked by the LCDCLK signal that the system interface unit generates by dividing the system clock. The LCDCLK signal is used to convert frame data to pixel format. The division factor depends on the type of display you are using. You should program LCDCLK using the following table. For information on the system clock and reset control register, refer to **Section 5 Clocks and Power Control**.

Table 18-1. LCDCLK Programming

BITS PER PIXEL/PANEL TYPE	LCD PANEL TYPE/DATA WIDTH				
	SINGLE-SCAN			DUAL-SCAN	
	12-BIT	8-BIT	4-BIT	8-BIT	4-BIT
1-Bit Black/White	—	F	F	F	F
2-Bit Grayscale	—	2 * F	F	2 * F	F
4-Bit Grayscale	—	2 * F	F	2 * F	F
4- or 8-Bit Color Passive	—	4 * F	2 * F	2 * F	—
4- or 8-Bit Color Active (TFT)	F	—	—	—	—

NOTE: F denotes the LCD panel clock frequency.

18.3.8 Contrast and Brightness Control

If you do not use frame rate control to handle contrast and brightness, then you will need external circuitry to give you more control for those panel inputs. Refer to your panel specifications to find out what you need for your panel.

18.3.9 The LCD Interface

The LCD interface provides drivers for the data, clock, and control signals to the LCD display. It consists of 12 data bits (LD[0:8], LCD_A, LCD_B, and LCD_C), a data clock (SHIFT/CLK), vertical and horizontal control (FRAME/VSYNC and LOAD/HSYNC), and panel control (LCD_AC/LOE). You can program the polarity of each of these signals. The LCD interface can be configured to support a variety of panels—single-scan, dual-scan, color, monochrome, passive, and active.

18.3.9.1 SINGLE-SCAN AND DUAL-SCAN PANELS. Some LCD panels split the display area into two horizontal halves that are scanned at the same time so that two lines are shifted and displayed simultaneously in each half. In this case, half of the data bus is used to drive the upper half of the screen and the other half is used to drive the lower half. Figure 18-10 illustrates single-scan and dual-scan LCD panels.

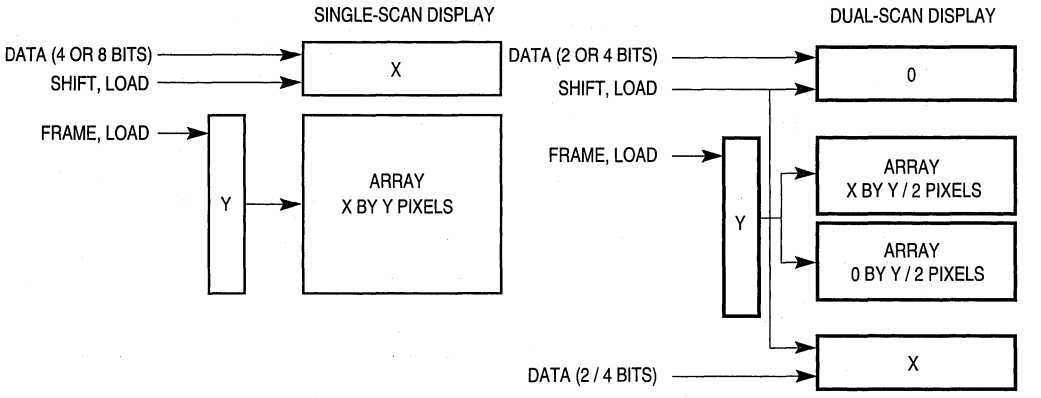


Figure 18-10. Single-Scan and Dual-Scan LCD Panels

18.3.9.2 PASSIVE INTERFACE. Passive LCD interfaces use the following signals. These signals have a programmable polarity. T_{cyc} is the cycle time of the LCD clock (SHIFT/CLK). T_{delay} is a circuit delay that is specified in **Section 22 Electrical Characteristics**.

- SHIFT/CLK—On the asserted edge of SHIFT, data is latched into the X shift register.
- FRAME/VSYNC—The FRAME signal initiates the frame by putting the Y pointer at the first row.
- LOAD/HSYNC—The LOAD signal transfers the contents of the shift register into the drive latches.
- LCD_AC/LOE—The LCD alternating current signal toggles every few frames to nullify any DC voltage. The toggle rate is programmable.
- LD—The width of this data bus is configured to 4 or 8 bits.

A general-purpose I/O can be used to output an integrated signal or pulse-width modulation (PWM) waveform and its duty cycle controls the RMS value of the voltage to the panel. The PWM signal is generated by one of the communication processor module timers. Refer to **Section 16.2.7 RISC Microcontroller Commands** for more details.

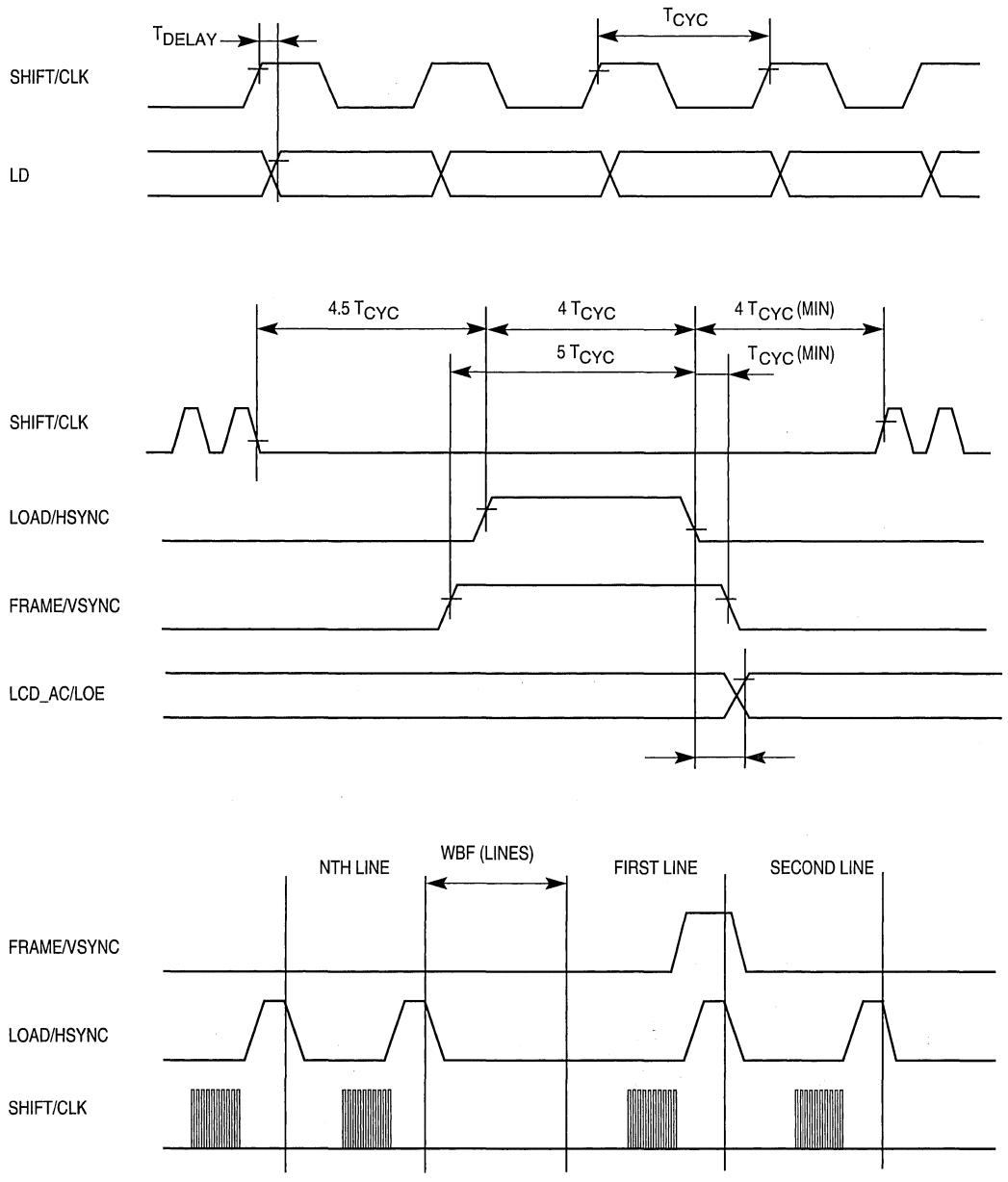


Figure 18-11. Passive Interface Timing Diagram

18.3.9.3 ACTIVE INTERFACE. Active (TFT) interfaces use the following signals. These signals have a programmable polarity. T_{cyc} is the cycle time of the LCD clock (SHIFT/CLK). T_{delay} is a circuit delay that is specified in **Section 22 Electrical Characteristics**. In Figure 18-12, the reference to 1-16 lines signifies that the time period depends on how the VPW field in the LCVCR is programmed. The reference to 0-1,023 lines signifies that the time period varies between 0 and 1,023 scan lines (WBF field in the LCVCR).

- SHIFT/CLK—When the LCD output enable signal is valid, data is latched on the asserted edge of CLK.
- FRAME/VSYNC—This vertical sync signal initiates a new frame.
- LOAD/HSYNC—This horizontal sync signal initiates a new line.
- LCD_AC/LOE—When the LCD output enable signal is valid it enables data to be shifted into the display. When it is disabled, the data is invalid and no data is transferred.
- LD—The LCD data bus represents 4-, 8-, or 12-bit data. For monochrome displays, 4- or 8-bit data is the same as passive interfaces and 12-bit data is used for color displays.

Use the following formulas to calculate the HSYNC and VSYNC cycles using the user-programmable parameters that are located in the LCD control registers.

VSYNC = (HSYNC × L) +WBF

HSYNC = SHIFT/CLK × (P ÷ LCDBW +12 + TWBL)

SHIFT/CLK = LCDCLK × F

LCDCLK = VCOOUT × LCD_div_factor (programmed)

Where:

L = Number of lines in panel (÷2 for dual-scan displays and +VPW in LCVCR for active).

WBF = Number of waits between frames.

P = Number of pixels per line in panel (×2 for dual-scan displays).

LCDBW = LCD bus width (4- or 8-bit for passive displays and 1 for active).

TWBL = Total number of waits between lines.

VCOOUT = SPLL output frequency.

LCD_div_factor = LCD_div_factor is programmed in the SCCR (DFLCD × DFALCD).

F = Inner rate factor that depends on your configuration:

3: For a four bits per pixel color passive display with an 8-bit LCD data bus width single-scan.

2: For a four bits per pixel color passive display with an 8-bit LCD data bus width dual-scan or with a 4-bit LCD data bus width single-scan.

1: For all other configurations.

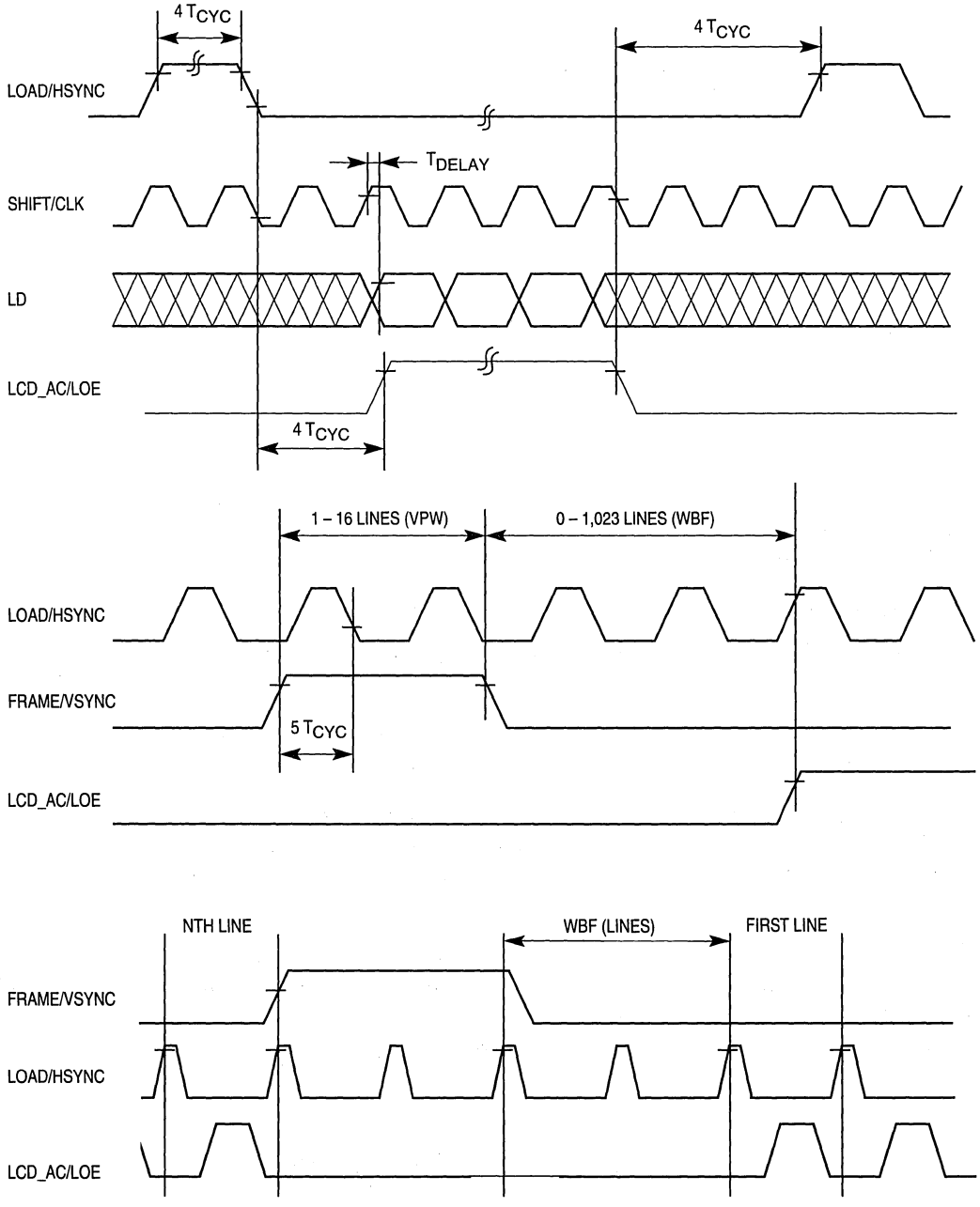


Figure 18-12. Active Interface Timing Diagram

18.3.9.4 ANALOG INTERFACE. The MPC823 has a digital interface, so you will need a DAC to connect the MPC823 to an analog panel.

18.3.10 System Considerations

When you are designing a system with the LCD controller, you should monitor the bus bandwidth used by the LCD subsystem and the maximum allowable bus latency. The configuration below uses the following parameters:

- GCLK2/CLKOUT—System clock frequency
- BNUM—Number of bursts per frame
- FRR—Frame refresh rate
- BPIX—Number of bits per pixel
- COL—Number of display columns
- ROW—Number of display rows
- MB—Number of system clocks per memory burst

You must configure the parallel to serial clock ratio between the system clock and the LCD serial data clock. The following example contains typical display characteristics for a full VGA panel (640 × 480) calculation:

- 640 × 480 = 307,200 pixels per screen
- (307,200 pixels per screen) × 4 bits per pixel = 1.228Mb per screen = 150Kb per screen
- 70Hz × 150Kb = 10.5Mb/s
- 70Hz is 14.3ms per frame: 14.3ms/307,200 pixels clocked 4 bits at a time = 186ns (approximately 5MHz) serial clock to the LCD drivers

The display serial clock is slightly faster than 5MHz because of the panel overhead. In most VGA displays, however, an 8-bit LCD serial data (dual-scan passive panel) is used and the display frequency is 3.76MHz. The CLKOUT to LCDCLK ratio should be between 4:1 and 5:1 for optimum system operation.

18.3.10.1 BUS BANDWIDTH. The bus bandwidth that the LCD controller uses depends on the display parameters (size, refresh rate), the memory system (number of clocks per burst), and the system clock frequency.

$$\text{BNUM} = \frac{\text{COL} \times \text{ROW} \times \text{BPIX}}{128}$$

$$\text{Bus Band Width} = \frac{\text{BNUM} \times \text{FRR} \times \text{MB}}{\text{SCLK}} \times 100\%$$

18.3.10.1.1 Example. The following example shows a monochrome full VGA (640×480) passive display, single-scan, 4-bit panel data bus. The LCD controller will provide 8-bit per pixel coding with a refresh rate of 80Hz. In this example, assume a display memory cycle burst timing of (2, 1, 1, 1) for a total of five cycles per burst.

SCLK = 25MHz

FRR = 90Hz

MB = 5

BPIX = 8

COL = 640

ROW = 480

BNUM = (COL × ROW × BPIX) ÷ 128

Bus Band Width = (BNUM × FRR × MB) ÷ SCLK

$$\text{Bus Band Width} = \frac{9600 \times 90 \times 5}{25 \times 10^6} \times 100\% = 17\%$$

18.3.10.2 BUS LATENCY. The maximum bus latency allowed in the system is given by:

$$\text{Max Latency} = \frac{MB \times SCLK}{BNUM \times FRR}$$

Typical example using the same data from above:

$$\text{Max Latency} = \frac{5 \times 25 \times 10^6}{9600 \times 90} = 145 \text{ System Clocks}$$

18.4 REGISTER MODEL

18.4.1 LCD Controller Configuration Register

The 32-bit, memory mapped, read/write LCD controller configuration register (LCCR) holds the mode and configuration parameters that you can use to operate your LCD panel.

LCCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	BNUM															EIEN
RESET	0															0
R/W	R/W															R/W
ADDR	(IMMR & 0xFFFF0000) + 0x840															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	IEN	IRQL			CLKP	OEP	HSP	VSP	DP	BPIX		LBW	SPLT	CLOR	TFT	PON
RESET	0	0			0	0	0	0	0	0		0	0	0	0	0
R/W	R/W	R/W			R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x842															

BNUM—Number of Bursts

This field contains the number of burst cycles required for one refresh cycle.

EIEN—Exception Interrupt Enable

0 = The underrun or bus error interrupt is disabled.
1 = The underrun or bus error interrupt is enabled.

IEN—Interrupt Enable

0 = The end-of-frame interrupt is disabled.
1 = The end-of-frame interrupt is enabled.

IRQL—Interrupt Request Level

This field contains the priority request level of the LCD controller's interrupt that is sent to the system interface unit. Refer to **Section 12.3.3 Programming the Interrupt Controller** for more information. This will generate an interrupt request level with a vector in the SIVEC register if enabled with the SIMASK register. Both EOF and the exception interrupts use the same request level.

CLKP—Clock Polarity

0 = The SHIFT/CLK pin polarity is active high.
1 = The SHIFT/CLK pin polarity is active low.

OEP—Output Enable Polarity

0 = The LCD_AC/LOE pin polarity is active high.
1 = The LCD_AC/LOE pin polarity is active low.

HSP—Horizontal Sync Polarity

0 = The LOAD/HSYNC pin polarity is active high.
1 = The LOAD/HSYNC pin polarity is active low.

VSP—Vertical Sync Polarity

0 = The FRAME/VSYNC pin polarity is active high.
1 = The FRAME/VSYNC pin polarity is active low.

DP—Data Polarity

0 = The LCD data (LD) pin polarity is active high.
1 = The LCD data (LD) pin polarity is active low.

BPIX—Bits Per Pixel

This field indicates the number of bits that represent one pixel in display memory.

00 = One bit per pixel.
01 = Two bits per pixel.
10 = Four bits per pixel.
11 = Eight bits per pixel.

LBW—LCD Bus Width

This field indicates the number of data bits that are output for every SHIFT/CLK. It is only valid for monochrome displays. For TFT displays, see the TFT bit below.

0 = Four bits per clock.
1 = Eight bits per clock.

SPLT—Split Display Mode

0 = The display is dual-scan (one row is displayed at a time).
1 = The display is single-scan (two rows are displayed at a time).

CLOR—Color Display

0 = The LCD panel is a monochrome display.
1 = The LCD panel is a color display.

TFT—TFT Display

When this bit is set, 12 bits of RGB (4 bits per color) data are provided on the LD data bus.

0 = The LCD panel is a passive display.
1 = The LCD panel is an active (TFT) display.

PON—Panel On

0 = The LCD controller operation is disabled.
1 = The LCD controller operation is enabled.

18.4.2 LCD Horizontal Control Register

The 32-bit, memory-mapped, read/write LCD horizontal control register (LCHCR) holds the panel horizontal pixel resolution and other configuration parameters.

LCHCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED							BO	AT			HPC				
RESET	0							0	0			0				
R/W	R/W							R/W	R/W			R/W				
ADDR	(IMMR & 0xFFFF0000) + 0x844															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	HPC						WBL									
RESET	0						0									
R/W	R/W						R/W									
ADDR	(IMMR & 0xFFFF0000) + 0x846															

Bits 0–6—Reserved

These bits are reserved and should be set to 0.

BO—Byte Order

0 = The DEC/Intel convention is used for byte ordering (swapped operation) and is also called little-endian byte ordering. The transmission order of bytes within a buffer word is reversed in comparison to the Motorola mode. This mode is supported only for 32-bit port size memory. Motorola byte ordering (normal operation) is also called big-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the most-significant byte of the buffer word contains data to be transmitted earlier than the least-significant byte of the same buffer word.
1 = PowerPC little-endian byte ordering. As data is transmitted onto the serial line from the data buffer, the least-significant byte of the buffer double-word contains data to be transmitted earlier than the most-significant byte of the same buffer double-word.

AT—Address Type

This field contains values that you want to appear on the AT pins when the associated SDMA channel accesses memory. AT0 is always driven to 1. Refer to **Section 13.4.7.3.4 Address Space Attributes** for address type descriptions.

HPC—Horizontal Pixel Count

This field specifies the number of pixels per line adjusted by panel type and bits per pixel. Use Table 18-2 to program the value for this field.

Table 18-2. Horizontal Pixel Count Programming

BITS PER PIXEL/PANEL TYPE	LCD PANEL TYPE/DATA WIDTH				
	SINGLE-SCAN			DUAL-SCAN	
	12 BIT	8-BIT	4-BIT	8-BIT	4-BIT
1-Bit Monochrome, 2-/4-Bit Grayscale	—	1/8 * H	1/4 * H	1/4 * H	1/2 * H
4-/8-Bit Color Passive	—	3/8 * H	3/4 * H	3/4 * H	—
4-/8-Bit Color Active (TFT)	H	—	—	—	—

NOTE: H indicates the number of pixels per line.

WBL—Wait Between Lines

To achieve the best display quality, this field can be used as an adjustable parameter to modify the resultant image. This field represents the wait period between lines, which are measured in SHIFT/CLK cycles. The total number of wait cycles are WBL+N, where:

N = 7 in configurations in which the panel type is a passive monochrome with a 4 bit data bus configured with two or four bits per pixel. BPIX = 01 or 10, CLOR = 0, TFT = 0, and LBW = 0 as defined in the LCCR.

N = 5 for all other configurations.

18.4.3 LCD Vertical Configuration Register

The 32-bit, memory-mapped, read/write LCD vertical control register (LCVCR) holds the panel vertical pixel resolution and other configuration parameters.

LCVCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	VPW				RESERVED			LCD_AC				VPC				
RESET	0				0			0				0				
R/W	R/W				R/W			R/W				R/W				
ADDR	(IMMR & 0xFFFF0000) + 0x848															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	VPC					RES	WBF									
RESET	0					0	0									
R/W	R/W					R/W	R/W									
ADDR	(IMMR & 0xFFFF0000) + 0x84A															

VPW—Vertical Sync Pulse Width (with active (TFT) panels only)

This field controls the width of the active FRAME/VSYNC signal with its value represented in line units. Programming this field to *n* causes FRAME/VSYNC to be active for *n* lines. This field is only valid for TFT displays and should be cleared for all others.

Bits 4–6 and 21—Reserved

These bits are reserved and should be set to 0.

LCD_AC—LCD AC Timing

This field specifies the number of frames that are displayed before the LCD_AC/LOE pin is toggled.

VPC—Vertical Pixel Count

This field specifies the number of lines per frame. Use Table 18-3 to program the value in this field.

Table 18-3. Vertical Pixel Count Programming

BITS PER PIXEL/PANEL TYPE	LCD PANEL TYPE/DATA WIDTH				
	SINGLE-SCAN			DUAL-SCAN	
	12-BIT	8-BIT	4-BIT	8-BIT	4-BIT
1-Bit Monochrome, 2-/4-Bit Grayscale	—	V	V	V/2	V/2
4-/8-Bit Color Passive	—	V	V	V/2	—
4-/8-Bit Color Active (TFT)	V	—	—	—	—

NOTE: V indicates the total number of lines on the panel.

WBF—Wait Between Frames

This field represents the wait period between frames, which is measured in lines.

18.4.4 LCD Frame Buffer A Start Address Register

The 32-bit LCD frame buffer A start address (LCFAA) register contains the start address of the frame buffer data that you want sent to your LCD panel. FIFO A is the destination for your frame buffer data. For single-scan panels, FIFO A concatenated with FIFO B is used to transfer data, so only the LCFAA register needs to be loaded. The DMA controller uses the buffer start address to initiate data transfers from display memory, which can be system memory or a dedicated display memory block. Because all LCD controller DMA bursts must be 16-byte aligned, the four least-significant bits of the address are not used. This register is read by the LCD controller at the start of each frame. Therefore, changing this register will not take effect until the WBF bit expires.

LCFAA

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	FAA															
RESET	—															
R/W	R/W															
ADDR	IMMR & 0xFFFF0000 + 0x850															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	FAA												X			
RESET	—												—			
R/W	R/W												R/W			
ADDR	(IMMR & 0xFFFF0000) + 0x852															

NOTE: X - "Don't Care" and — = Undefined.

FAA—FIFO A Address

This field designates the start address in display or system memory where LCD panel data resides.

18.4.5 LCD Frame Buffer B Start Address Register

The 32-bit LCD frame buffer B start address (LCFBA) register contains the start address of the frame buffer data that you want to send to your LCD dual-scan panel (lower half). FIFO B is is the destination for your frame buffer data to be passed to the lower half of the panel. Notice that for single-scan panels FIFO B is concatenated with FIFO A to transfer data, so only the LCFAA register needs to be loaded. However, for dual-scan panels, the LCFBA register must be set. For these dual-scan panels, the DMA controller uses the buffer B start address to initiate data transfers from display memory (system memory or a dedicated display memory block) to FIFO B. Because all LCD controller DMA bursts must be 16-byte aligned, the four least-significant bits of the address are not used. This register is read by the LCD controller at the start of each frame. Therefore, changing this register will not take effect until the WBF bit expires.

LCFBA

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	FBA															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x854															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	FBA												X			
RESET	—												—			
R/W	R/W												R/W			
ADDR	IMMR & 0xFFFF0000 + 0x856															

NOTE: X - "Don't Care" and — = Undefined.

FBA—FIFO B Address

This field designates the start address in display or system memory where the LCD panel data resides. The data retrieved is for the lower half of a dual-scan panel and passes through FIFO B.

18.4.6 LCD Status Register

The 8-bit memory-mapped LCD status register (LCSR) is used to report certain events to the core. When an event is recognized, the LCD controller sets its corresponding bit in this register, regardless of the corresponding enable bit, which is located in the LCCR. A bit is cleared by writing a 1 (writing a 0 has no effect) and more than one bit can be cleared at a time.

LCSR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED					BERR	UN	EOF
RESET	0					0	0	0
R/W	R/W					R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x858							

Bits 0–4—Reserved

These bits are reserved and should be set to 0.

BERR—Bus Error

This status bit is set if a display memory read cycle by the LCD controller FIFO is abnormally terminated. If the EIEN bit is set in the LCCR, then an interrupt is generated to the system interface unit at the level specified in the IRQL field of LCCR.

UN—Underrun

When this bit is set, it indicates that a FIFO underrun condition has been detected. An underrun condition occurs when the LCD controller is empty before a frame is completed. If the EIEN bit is set in the LCCR, then an interrupt is generated to the system interface unit at the level specified in the IRQL field of LCCR.

EOF—End Of Frame

This status bit is set when a frame is completed and if the IEN bit in the LCCR is enabled. Then an interrupt is generated to the system interface unit at the level specified in the IRQL field of LCCR.

18.4.7 Color RAM Operation Modes

The color RAM contains 256 entries that are each 16 bits wide. It is located in the dual-port RAM and is not initialized at reset. Your LCD panel and its required mode will dictate how the display or system memory and the color RAM is configured.

Note: Before programming the color RAM, you must program the LCCR to the specific data coding, number of bits per pixel, color or monochrome, etc.

18.4.7.1 ONE BIT PER PIXEL MONOCHROME MODE. When you are using this mode (TFT=0, BPIX=00, and CLOR=0), configure color RAM using the following pattern. You should program the first 16 entries of the color RAM to be transparent.

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED												GLC			
RESET	—												—			
R/W	R/W												R/W			
ADDR	(IMMR & 0xFFFF0000) + (DPR) 0xE00—0xE1F															

NOTE: — = Undefined.

Bits 0–11—Reserved

These bits are reserved and should be set to 0.

GLC—Grayscale Level Code (Monochrome Display)

This field is a 4-bit code that represents the grayscale level. It must be programmed to the value of the entry number. For entry 0 GLC = 0000, for entry 1 GLC = 0001, and for entry 2 GLC = 0010 as shown below.

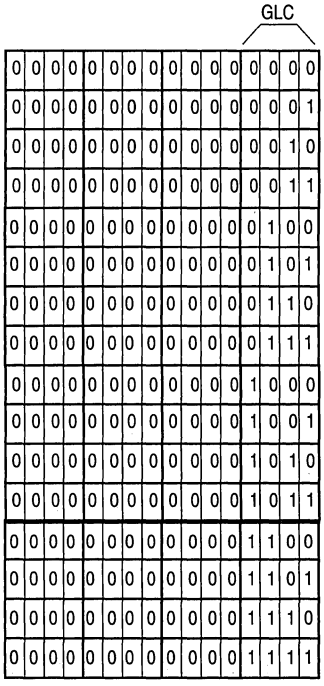


Figure 18-13. Color RAM Transparent Translation for One-Bit Per Pixel Mode

18.4.7.2 TWO BITS PER PIXEL GRAYSCALE MODE. In two bits per pixel grayscale mode, the LCD controller provides four possible shades to be loaded into each pixel of the LCD panel. The two bits of data are provided by each display memory word that is accessed by the DMA controller. This value will then index into the color RAM at addresses 1, 3, 5, and 7. You can provide any four of 16 possible shades by loading the color RAM addresses shown below.

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED								GLCB				GLCA			
RESET	—								—				—			
R/W	R/W								R/W				R/W			
ADDR	(IMMR & 0xFFFF0000) + (DPR) 0xE00—0xE07															

NOTE: — = Undefined.

Bits 0–7—Reserved

These bits are reserved and should be set to 0.

GLCA/GLCB—Grayscale Level Code A and B

This field is a 4-bit code that represents the grayscale level for a given pixel code. GLCA and GLCB should be programmed to the same value. The 4-bit code should be programmed in one of the following ways:

- Program address 1 of the color RAM with the grayscale level code that corresponds to pixel code 00 that was retrieved from display memory.
- Program address 3 of the color RAM with the grayscale level code that corresponds to pixel code 01 that was retrieved from display memory.
- Program address 5 of the color RAM with the grayscale level code that corresponds to pixel code 10 that was retrieved from display memory.
- Program address 7 of the color RAM with the grayscale level code that corresponds to pixel code 11 that was retrieved from display memory.

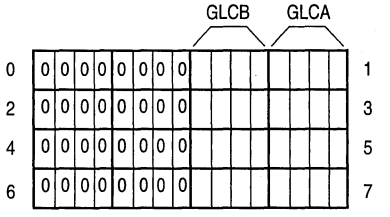


Figure 18-14. Color RAM Entries for Two Bits Per Pixel Mode

18.4.7.3 FOUR BITS PER PIXEL GRAYSCALE MODE. In four bits per pixel grayscale mode, the LCD controller provides 16 possible shades to be loaded into each pixel of the LCD panel. The four bits of data are provided by each display memory word that is accessed by the DMA controller. This value will then index into the first 16 odd addresses of the color RAM. You can provide any of the 16 possible shades by loading the color RAM addresses shown below. See Figure 18-8 for more information.

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED								GLCB				GLCA			
RESET	—								—				—			
R/W	R/W								R/W				R/W			
ADDR	(IMMR &0xFFFF0000) + (DPR) 0xE00—0xFFF															

NOTE: — = Undefined.

Bits 0–7—Reserved

These bits are reserved and should be set to 0.

GLCA/GLCB—Grayscale Level Code A and B

This field is a 4-bit code that represents any one of 16 possible grayscale levels for a given pixel code. This grayscale level is then used to represent the appropriate shade for a given pixel and is passed to the appropriate panel bit. GLCA and GLCB should be programmed to the same value. The color RAM to be programmed is shown below.

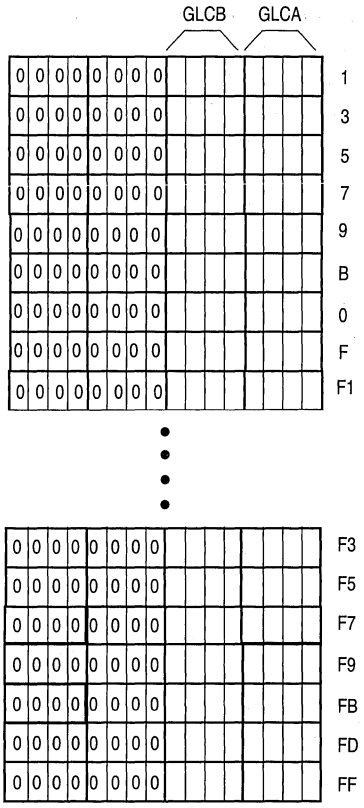


Figure 18-15. Color RAM Entries for Four Bits Per Pixel (Grayscale)

18.4.7.4 PASSIVE FOUR AND EIGHT BITS PER PIXEL COLOR MODE. This mode operates the same way for four or eight bits per pixel and the color RAM is loaded similarly. You provide a 4- or 8- bit data word that the DMA controller retrieves from display memory and that value is indexed into the color RAM as shown below. For four bit per pixel mode, you can choose 1 of 16 colors from the 4,096 possible colors. For eight bits per pixel mode, you can choose one of 256 colors from the 4,096 possible colors. See Figure 18-9 for more information.

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED				R				G				B			
RESET	—				—				—				—			
R/W	R/W				R/W				R/W				R/W			
ADDR	(IMMR & 0xFFFF0000) + (DPR) 0xE00—0xE1F (4-BIT) OR 0xE00—0xFFF (8-BIT)															

NOTE: — = Undefined.

Bits 0–3—Reserved

These bits are reserved and should be set to 0.

R—Red Level

This field is a 4-bit code that represents the red color level. The frame rate control dictates when this color pixel is on or off.

G—Green Level

This field is a 4-bit code that represents the green color level. The frame rate control dictates when this color pixel is on or off.

B—Blue Level

This field is a 4-bit code that represents the blue color level. The frame rate control dictates when this color pixel is on or off.

18.4.7.5 ACTIVE FOUR AND EIGHT BITS PER PIXEL COLOR MODE. This mode operates the same way for four or eight bits per pixel and the color RAM is loaded similarly. You provide a 4- or 8- bit data word that the DMA controller retrieves from display memory and that value is indexed into the color RAM as shown below. For four bits per pixel mode, you can choose 1 of 16 out of a total of 4,096 possible colors. For eight bits per pixel mode, you can choose one of 256 out of the 4,096 possible colors. See Figure 18-9 for more information. For active (TFT) panels, each bit of data for each red, green, or blue value read from the color RAM is passed directly to the LCD data bus. These bits follow a particular sequence—LD0 (MSB), LD1, LD2, LCD_A, LD3, LD4, LD5, LCD_B, LD6, LD7, LD8, and LCD_C (LSB).

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RESERVED				R				G				B			
RESET	—				—				—				—			
R/W																
ADDR	(IMMR & 0xFFFF0000) + (DPR) 0xE00—0xE1F (4-BIT) OR 0xE00—0xFFF (8-BIT)															

NOTE: — = Undefined.

Bits 0–3, 7, 11, and 15—Reserved

These bits are reserved and should be set to 0.

R—Red Level

This field is a 4-bit code that represents the red color level and is passed directly to the LCD interface.

G—Green Level

This field is a 4-bit code that represents the green color level and is passed directly to the LCD interface

B—Blue Level

This field is a 4-bit code that represents the blue color level and is passed directly to the LCD interface

18.4.8 LCD Panel Connection Examples

Some panels connect differently than the MPC823. These panels are shown as examples in Table 18-4 and they may or may not be currently available.

Table 18-4. LCD Panel Connection

MPC823 SIGNAL (PORT PIN)	DISPLAY TYPES					
	SHARP LQ9D021 ACTIVE COLOR	SHARP LM4801F PASSIVE MONOCHROME	SHARP LM64P839 PASSIVE MONOCHROME DUAL-SCAN	HITACHI LMG7211URFR PASSIVE MONOCHROME	SHARP LM32K07 PASSIVE MONOCHROME	SHARP LQ10D131 COLOR ACTIVE
SHIFT/CLK (PD3)	CK-P[10]	CP2-P[3]	CP2-P[3]	CL2-P[3]	CP2-P [10]	CK-P[10]
LOAD/HSYNC (PD4)	HSYNC-P[6]	CP1-P[2]	CP1-P[2]	CL1-P[2]	CP1-P [11]	HSYNC-P[6]
FRAME/VSYNC (PD5)	VSYNC-P[4]	S-P [1]	S-P[1]	FRAME-P[1]	S-P [12]	VSYNC-P[4]
LCD_AC/LOE (PD6)	ENAB-P [28]	—	—	—	—	ENAB-P[28]
LD0 (PD7)	R2-P [7]	—	—	—	—	R3-P[7]
LD1 (PD8)	R1-P [5]	—	DU3-P [11]	—	—	R2-P[5]
LD2 (PD9)	R0-P [3]	—	DU2-P [10]	—	—	R1-P[3]
LCD_A (PB31)	—	—	—	—	—	R0-P[1]
LD3 (PD10)	G2-P [19]	—	DU1-P[9]	—	—	G3-P[19]
LD4 (PD11)	G1-P [17]	—	DU0-P[8]	—	—	G2-P[17]
LD5 (PD12)	G0-P [13]	DU3-P [11]	DL3-P [15]	D3-P [11]	D3-P [3]	G1-P[13]
LCD_B (PB15)	—	—	—	—	—	G0-P[P11]
LD6 (PD13)	B2-P [29]	DU2-P [10]	DL2-P [14]	D2-P [10]	D2-P [4]	B3-P[31]
LD7 (PD14)	B1-P [27]	DU1-P[9]	DL1-P [13]	D1-P [9]	D1-P [5]	B2-P[29]
LD8 (PD15)	B0-P [25]	DU0-P[8]	DL0-P [15]	D0-P [8]	D0-P [6]	B1-P[25]
LCD_C (PB17)	—	—	—	—	—	B0-P[23]

SECTION 19
VIDEO CONTROLLER

The MPC823 contains an on-chip video controller that can be used to drive a digital TFT LCD panel or an analog NTSC/PAL display (which needs an external video encoder). It uses the same I/O pins as the LCD controller, which means you can only be using one or the other at a time. The video controller has features that give you more flexibility when preparing data for a video display.

The video controller uses a frame buffer, also called a display buffer, that is stored in system memory. The data in the display buffer represents pixel components (bytes), whether it is RGB or YC_rC_b. You are responsible for preprocessing data as it is stored in the display buffer. The video controller uses a dedicated DMA channel to read the data from the display buffer and drive it to the video interface. It also generates the required timing and control signals (horizontal sync, vertical sync, field, and blanking). A typical MPC823 video system is illustrated in Figure 19-1.

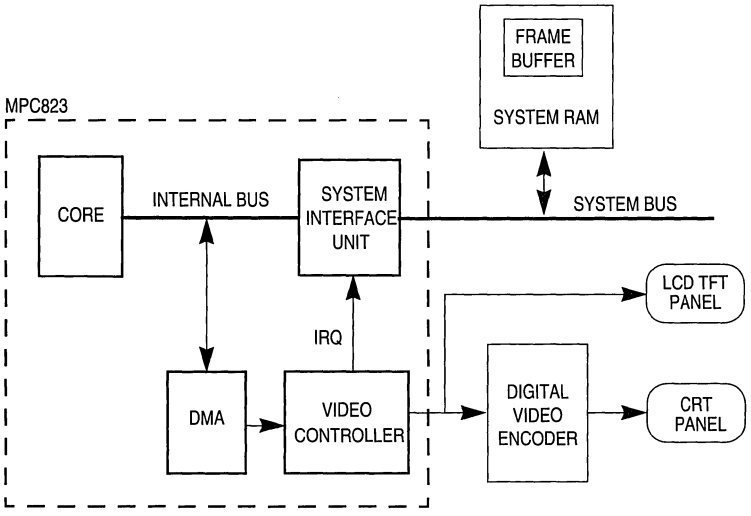


Figure 19-1. Typical MPC823 Video System

19.1 FEATURES

The following list summarizes the features of the video controller:

- Supports Digital TFT LCD Panels and Analog NTSC/PAL Displays
- Sequential RGB, 4:4:4, and 4:2:2 YC_rC_b (CCIR 601) Digital Component Video Formats
- CCIR-656 Compatible 8-Bit Interface Port
- Programmable Control for Horizontal Sync, Vertical Sync, Field, Blanking, Polarity, and Timing Generation with Half-Clock Resolution
- Supports Interlace/Noninterlace Scanning Methods
- Programmable Display Active Area
- Programmable Background Color for Inactive Areas
- Smooth Switching Between Two Picture Formats
- Supports Hardware Pan/Scroll Options in a Zoomed Buffer
- Glueless Interface to Most Digital Video Encoders
- Burst Read DMA Cycles Are Used for Maximum Bus Performance
- End-of-Frame and Bus Exception Interrupt Generation

19.2 OPERATION

The video controller consists of a register set, DMA controller with FIFOs, and a video control RAM array, as shown in Figure 19-2. The video controller RAM array provides the proper sequencing and control signal generation needed to synchronize the datastream through the FIFOs. The video controller, a standalone module, is programmed using a set of configuration registers. Once the registers have been configured and the video controller is enabled, the DMA controller initiates burst read cycles to display memory.

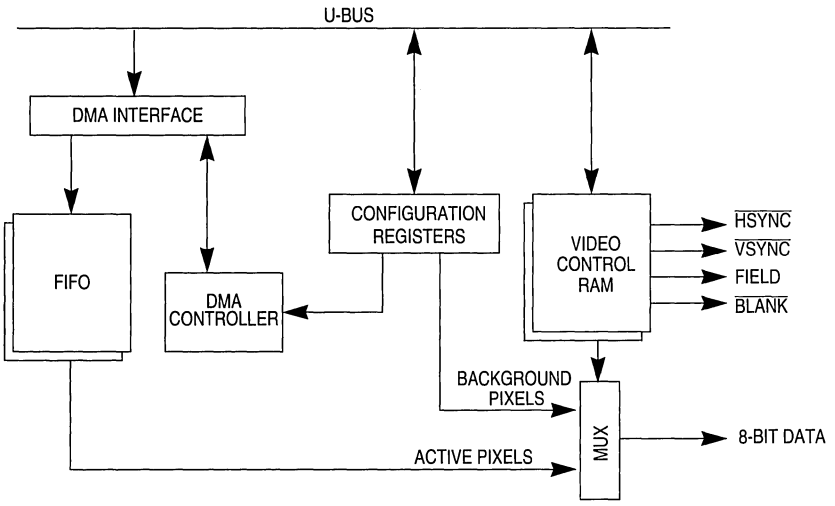


Figure 19-2. Video Controller Block Diagram

19.2.1 The Video Controller Clock

The video controller master clock source can either be the LCDCLK, which is generated by the system interface unit, or the external video clock signal (CLK). Refer to **Section 5.2.1 System Clock and Reset Control Register** and **Section 5.3.4.4 The LCD Clocks** for more information. LCDCLK is derived from the SPLL. If the external video clock is enabled by setting the CSRC bit in the VCCR, you must provide an external video clock (CLK) at the port D input.

Note: If an external video clock (CLK) is used, then the ratio between that external clock and GCLK1 must not be greater than 1.25:1. For example, if your system clock is 50MHz, then the CLK input cannot exceed 62.5MHz.

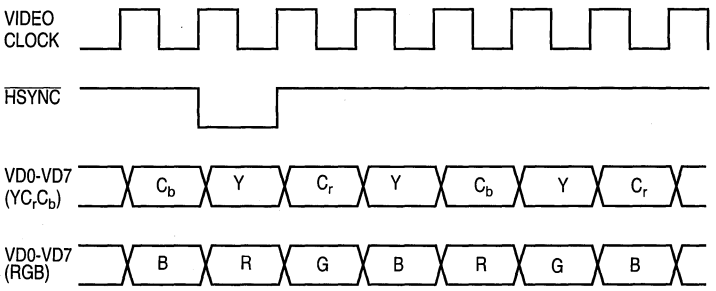


Figure 19-3. Output Timing Example

19.2.2 FIFO and DMA Control

The video controller FIFO consists of 24 32-bit entries. The DMA control block handles all data transfers to the FIFOs and keeps them filled. At the appropriate time, each FIFO is filled by the DMA control block in 16-byte transfers. The frame buffer address must be 16-byte aligned. When the video controller is enabled, video data transfers to the panel start after five DMA burst accesses have filled the FIFOs. See **Section 16.5.1 SDMA Bus Arbitration and Transfers** for the proper arbitration configuration between the video controller DMA and SDMA. The FIFO has two sets of control registers (0 and 1) associated with the video RAM arrays (RAM_0 or RAM_1 appropriately). If there are problems with the screen blanking when your caches are on, set the SDCR to 0x40.

The DMA control supports both interlace and noninterlace (progressive) scanning schemes. In interlace mode, the DMA fetches the video components of odd lines from display frame buffer A followed by the even lines from display frame buffer B. In noninterlace mode, the lines are fetched sequentially from buffer A. No matter which mode you use, the video components of a line must be stored in an integer number of bursts.

If the FIFO underruns during a frame, the video controller forces background video on the screen until synchronization between the FIFO and video control RAM array is regained. Synchronization means the FIFO and the video control RAM array are ready to display a frame from its beginning. At that point, the video controller starts reprocessing the frame at the beginning of the video control RAM array.

19.2.3 Image Sizes

The video controller can be used to display an image that is smaller than the size of the display. The area that is used to display the image is defined as the "active display area" and its video components are taken from the frame buffer. The inactive area is driven with a single user-programmable default background color. The video controller changes smoothly between two image sizes without disturbing the video timings. A display format is defined by the pattern in the RAM array (set RAM_0 or RAM_1) and the appropriate FIFO control register set (set_0 or set_1). The active RAM set contains the pattern associated with the currently displayed image and the other RAM set contains the pattern associated with the second image.

After you program the inactive RAM set and corresponding FIFO control register set, change the ASEL bit in the VCCR to the value of the inactive set. The video controller will switch between the currently active set and the inactive set at the boundary of the frame. The CAS bit in the VSR reflects the currently active set.

You can also force background video frames without disturbing the video timings by setting the BD bit in the VCCR. During this time, the FIFO is flushed and allows you to reprogram the current FIFO buffer address pointer registers. You can also switch between the currently active and inactive sets during this period. After clearing the BD bit, the video controller will continue operating according to the current RAM set and FIFO control register set.

Note: If the images contain less than 24 words of data, the video controller will not have time to change images smoothly. You should use the force background video method described above to ensure correct operation.

19.3 REGISTER MODEL

19.3.1 Video Controller Configuration Register

The 16-bit, memory-mapped, read/write video controller configuration register (VCCR) contains the mode and configuration bits for the video controller.

VCCR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	DDT	DP	DPF	RES	IEN	EIEN	IRQL			BO	AT			RES	CSRC	VON
RESET	0	0	0	0	0	0	0			0	0			0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W			R/W	R/W			R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x800															

DDT—Data Drive Timing

This bit determines when the data changes.

0 = The video controller drives new data on the rising edge of the video clock.
1 = The video controller drives new data on the falling edge of the video clock.

DP—Data Polarity

0 = The data polarity is active high.
1 = The data polarity is active low.

DPF—Default Pixels Format

This bit defines the format of the background video components.

0 = BGND1, BGND2, BGND3 values will be used for the inactive area (BGND1, BGND2, BGND3, BGND1, BGND2, BGND3,....). This setting is used for RGB encoding.
1 = BGND1, BGND2, BGND3, BGND4 values will be used for the inactive area (BGND1, BGND2, BGND3, BGND4, BGND1, BGND2, BGND3, BGND4,...). This setting is used for YC_rC_b format encoding ($Y_1C_rY_2C_b$).

Bits 3 and 13—Reserved

These bits are reserved and must be set to 0.

IEN—Interrupt Enable

0 = The end-of-frame (EOF) interrupt is disabled.
1 = The end-of-frame (EOF) interrupt is enabled.

EIEN—Exception Interrupt Enable

0 = The underrun/bus error interrupt is disabled.
1 = The underrun/bus error interrupt is enabled.

IRQL—Interrupt Request Level

This field contains the priority request level of the video controller's interrupt that is sent to the system interface unit. Refer to **Section 12.3.3 Programming the Interrupt Controller** for more information. This will generate an interrupt request level with a vector in the SIVEC register if enabled with the SIMASK register. Both EOF and the exception interrupts use the same request level. 000 is the highest priority level and 111 is the lowest.

BO—Byte Order

0 = PowerPC little-endian byte order.
1 = Big- or little-endian byte order.

AT—Address Type

This field contains the value that you want to appear on the AT pins when the associated SDMA channel accesses memory. AT0 will always be driven to 1. Refer to **Section 13.4.7.3.4 Address Space Attributes** for address type descriptions.

CSRC—Clock Source

This bit controls the clock source to the video controller.

0 = The video controller source is LCDCLK.
1 = The video controller source is the SHIFT/CLK/CLK pin (PD3).

VON—Video Controller On

0 = Video controller operation is disabled.
1 = Video controller operation is enabled.

19.3.2 Video Status Register

The 8-bit memory-mapped video status register (VSR) is used to report certain events to the core. When an event is recognized, the video controller sets its corresponding bit in the video status register, regardless of the corresponding mask bit. The video status register can be read at any time. A bit is cleared by writing a one (writing a zero has no effect). More than one bit may be cleared at a time.

VSR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED	CAS	RESERVED			BERR	UN	EOF
RESET	0	0	0			0	0	0
R/W	R/W	R	R/W			R/W	R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x804							

Bits 0 and 2–4—Reserved

These bits are reserved and should be set to 0.

CAS—Current Active Set

This read-only bit indicates the currently active RAM array and FIFO control register set.

0 = RAM_0 and FIFO register set_0 are currently active.
1 = RAM_1 and FIFO register set_1 are currently active.

BERR—Bus Error

This status bit is set if a read cycle associated with the video controller was terminated by the assertion of a $\overline{\text{TEA}}$ signal. A maskable interrupt is generated to the core for as long as this bit is set.

UN—Underrun

This bit indicates that a FIFO underrun condition has been detected. An underrun condition occurs when the video controller FIFO is empty before a frame is completed. If the EIEN bit is set in the VCCR, then an interrupt is generated to the system interface unit at the level specified in the IRQL field of VCCR.

EOF—End Of Frame

This status bit is set when a RAM line, with the INT bit set, is executed. If the RAM line is repeated for more than one clock cycle, EOF will only be set at the last cycle of execution. This bit is typically used to mark the completion of a frame. If the IEN bit in the VCCR is enabled, an interrupt is generated to the system interface unit at the level specified in the IRQL field of VCCR.

19.3.3 Video Command Register

The 8-bit video command register (VCMR) is used to control the display format.

VCMR

BIT	0	1	2	3	4	5	6	7
FIELD	RESERVED						ASEL	BD
RESET	0						0	0
R/W	R/W						R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0x806							

Bits 0–5—Reserved

These bits are reserved and should be set to 0.

ASEL—Active Set Select

This bit selects one RAM array and FIFO control register set to be active for the next frame. The current set selection is reflected in the CAS bit of the VSR.

0 = Selects RAM_0 and FIFO register set_0 as the active set.
1 = Selects RAM_1 and FIFO register set_1 as the active set.

Note: Once the ASEL bit is changed, you cannot access video RAM until the CAS bit in the VSR reflects the change.

BD—Blank Display

When set, this bit forces the background video to be displayed and flushes the current FIFO.

0 = Display the image from the frame buffer.
1 = Force background video and flush FIFO.

19.3.4 Video Background Color Buffer Register

The 32-bit, read/write video background color buffer (VBCB) register holds components for the background video. It is used as the source of video data for the inactive area of the display, according to the DPF bit in the VCCR. When you are using the ADV7176, this register should be programmed as C_bYC_rY.

VBCB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	BGND1								BGND2							
RESET	0								0							
R/W	R/W								R/W							
ADDR	(IMMR & 0xFFFF0000) + 0x808															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	BGND3								BGND4							
RESET	0								0							
R/W	R/W								R/W							
ADDR	(IMMR & 0xFFFF0000) + 0x80A															

BGNDx—Background Color Component 1–4

This field represents the color component based on the RGB or YC_rC_b format.

19.3.5 Video Frame Configuration Register (Set 0)

The 32-bit, memory mapped, read/write video frame configuration register set 0 (VFCR0) holds the display horizontal and vertical size, as well as the gap between two sequential lines.

VFCR0

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	SFB0	RESERVED		VPC0										GAP0		
RESET	0	0		0										0		
R/W	R/W	R/W		R/W										R/W		
ADDR	(IMMR & 0xFFFF0000) + 0x810															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	GAP0								NBPL0							
RESET	0								0							
R/W	R/W								R/W							
ADDR	(IMMR & 0xFFFF0000) + 0x812															

SFB0—Single Frame Buffer 0

This bit controls whether the video controller displays an image from a single frame buffer (A) or from both frame buffers (A and B).

0 = Frame B is valid.
1 = Frame B is not valid.

Bits 1–2—Reserved

These bits are reserved and should be set to 0.

VPC0—Vertical Pixel Count 0

This field defines the number of lines for a field.

Note: The value of the VPC0 field should be non-zero or an error will occur.

GAP0—Gap 0

This field defines the gap in memory between the end of a line and the beginning of the next line in full burst units. For regular noninterlace mode, this field is set to 0. For regular interlace mode, it is set to the value in the NBPL0 field. For example, hardware pan/scroll options in a zoomed buffer can be implemented by using the GAP0 field with an appropriate field buffer start address. For example, with 720 pixels in YC_rC_b format, GAP0 should be 0x5A.

NBPL0—Number of Bursts per Line 0

This field defines the number of bursts per line.

Note: The value of the NBPL0 field should be non-zero or an error will occur.

19.3.6 Video Frame Buffer A Start Address Register (Set 0)

Th 32-bit video frame buffer A start address register set 0 (VFAA0) holds the start address of the set_0 odd field. Since all bursts must be 16-byte aligned, this register does not use the four least-significant bits of the address.

VFAA0

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	FAA0															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x814															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	FAA0												X			
RESET	—												—			
R/W	R/W												R/W			
ADDR	(IMMR & 0xFFFF0000) + 0x816															

NOTE: X = "Don't Care" and — = Undefined.

FAA0—Frame Buffer A Start Address for Set 0

This field designates the start address of the frame buffer A set 0 in system memory.

19.3.7 Video Frame Buffer B Start Address Register (Set 0)

The 32-bit video frame buffer B start address register set 0 (VFBA0) holds the start address of the set_0 even field. Since all bursts must be 16-byte aligned, this register does not use the four least-significant bits of the address.

VFBA0

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	FBA0															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x818															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	FBA0												X			
RESET	—												—			
R/W	R/W												R/W			
ADDR	(IMMR & 0xFFFF0000) + 0x81A															

NOTE: X = "Don't Care" and — = Undefined.

FBA0—Frame Buffer B Start Address for Set 0

This field designates the start address of the frame buffer B set 0 in system memory.

19.3.8 Video Frame Configuration Register (Set 1)

The video frame configuration register set 1 (VFCR1) has the same structure as VFCR0, except it belongs to set 1 and VFCR0 belongs to set 0. The value of the VPC1 and NBPL1 fields should be non-zero or an error will occur.

VFCR1

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	SFB1	RESERVED		VPC1										GAP1		
RESET	0	0		0										0		
R/W	R/W	R/W		R/W										R/W		
ADDR	(IMMR & 0xFFFF0000) + 0x81C															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	GAP1								NBPL1							
RESET	0								0							
R/W	R/W								R/W							
ADDR	(IMMR & 0xFFFF0000) + 0x81E															

SFB1—Single Frame Buffer 1

This bit controls whether the video controller displays an image from a single frame buffer (A) or from both frame buffers (A and B).

0 = Frame B is valid.
1 = Frame B is not valid.

Bits 1–2—Reserved

These bits are reserved and should be set to 0.

VPC1—Vertical Pixel Count 1

This field defines the number of lines for a field.

GAP1—Gap 1

This field defines the gap in the memory between the end of a line and the beginning of the next line in full burst units. For regular noninterlace mode, this field is set to 0. For regular interlace mode, it is set to the value in the NBPL1 field. For example, hardware pan/scroll options in a zoomed buffer can be implemented by using the GAP1 field with an appropriate field buffer start address.

NBPL1—Number of Bursts per Line 1

This field defines the number of bursts per line.

19.3.9 Video Frame Buffer A Start Address Register (Set 1)

The 32-bit video frame buffer A start address register set 1 (VFAA1) holds the start address of the set_1 odd field. Since all bursts are required to be 16-byte aligned, this register does not use the four least-significant bits of the address.

VFAA1

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	FAA1															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x820															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	FAA1												X			
RESET	—												—			
R/W	R/W												R/W			
ADDR	(IMMR & 0xFFFF0000) + 0x822															

NOTE: X = - "Don't Care" and — = Undefined.

FAA1—Frame Buffer A Start Address for Set 1

This field designates the start address of the frame buffer A set 0 in system memory.

19.3.10 Video Frame Buffer B Start Address Register (Set 1)

The 32-bit video frame buffer B start address register set 1 (VFBA1) holds the start address of the set_1 even field. Since all bursts are required to be 16-byte aligned, this register does not use the four least-significant bits of the address.

VFBA1

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	FBA1															
RESET	—															
R/W	R/W															
ADDR	(IMMR & 0xFFFF0000) + 0x824															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	FBA1												X			
RESET	—												—			
R/W	R/W												R/W			
ADDR	(IMMR & 0xFFFF0000) + 0x826															

NOTE: X = "Don't Care" and — = Undefined.

FBA1—Frame Buffer B Start Address for Set 1

This field designates the start address of the frame buffer B set 0 in system memory.

19.4 VIDEO CONTROLLER RAM ARRAY

The video controller state machine controls data that is shifted out to the video port as well as the timing patterns of the $\overline{\text{HSYNC}}$, $\overline{\text{VSYNC}}$, FIELD, and $\overline{\text{BLANK}}$ signals. The video RAM consist of two RAM arrays—RAM_0 and RAM_1—that contain 64 32-bit entries. At any given time, one RAM array actively drives the panel and controls video controller operation, while the other is inactive but modifiable. You can switch between the two RAMs at any time, but your change will not take effect until the end of the frame. The CAS bit in the VSR reflects the RAM that is active. An entry in the active RAM is read each video clock and specifies the state of the video port signals for the next CNT video clocks. CNT is specified by a special field within the entry. The next entry is read and used after the CNT clock periods of the previous entry. A few entries can be repeated in a loop to generate a repetitive pattern. Read/write operations are always directed to the inactive RAM array and can be performed anytime the active RAM controls the video controller and display operation. Since you can only access one RAM array (the inactive one) at a time, both RAMs are mapped to the same address space. The RAM arrays are not initialized after power-on and any access to the RAM array is discouraged while the video controller clock inputs are not operating. The video RAM array block diagram is illustrated in Figure 19-4.

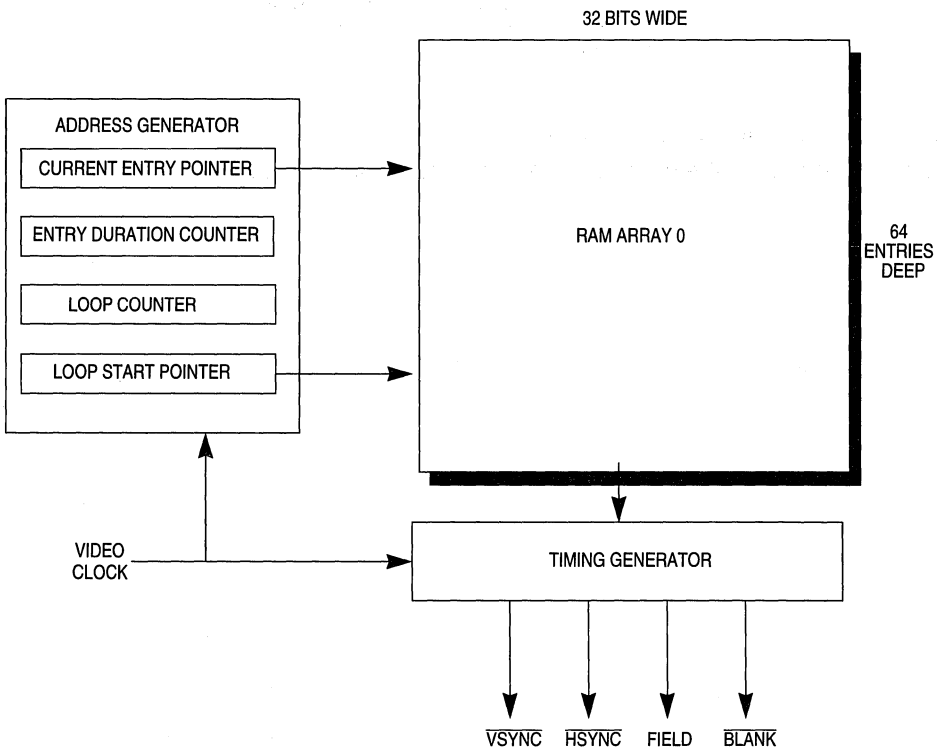


Figure 19-4. Video RAM Array Block Diagram

19.4.1 Video RAM Word Format

The video RAM word specifies the timing of all external signals controlled by the video controller.

VIDEO RAM WORD

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	HR	HF	VR	VF	FR	FF	BR	BF	RESERVED						VDS	
RESET	—	—	—	—	—	—	—	—	—						—	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W						R/W	
ADDR	(IMMR & 0xFFFF0000) + 0xB00–0xBFE															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	INT	RESERVED		LCYC/CNT											LP	LST
RESET	—	—		—											—	—
R/W	R/W	R/W		R/W											R/W	R/W
ADDR	(IMMR & 0xFFFF0000) + 0xB00–0xBFF															

NOTE: — = Undefined.

HR—Horizontal Sync on Rising Edge of the Clock

0 = The value of the $\overline{\text{HSYNC}}$ signal will be 0 after the rising edge of the clock.
1 = The value of the $\overline{\text{HSYNC}}$ signal will be 1 after the rising edge of the clock.

HF—Horizontal Sync on Falling Edge of the Clock

0 = The value of the $\overline{\text{HSYNC}}$ signal will be 0 after the falling edge of the clock.
1 = The value of the $\overline{\text{HSYNC}}$ signal will be 1 after the falling edge of the clock.

VR—Vertical Sync on Rising Edge of the Clock

0 = The value of the $\overline{\text{VSYNC}}$ signal will be 0 after the rising edge of the clock.
1 = The value of the $\overline{\text{VSYNC}}$ signal will be 1 after the rising edge of the clock.

VF—Vertical Sync on Falling Edge of the Clock

0 = The value of the $\overline{\text{VSYNC}}$ signal will be 0 after the falling edge of the clock.
1 = The value of the $\overline{\text{VSYNC}}$ signal will be 1 after the falling edge of the clock.

FR—Field on Rising Edge of the Clock

0 = The value of the FIELD signal will be 0 after the rising edge of the clock.
1 = The value of the FIELD signal will be 1 after the rising edge of the clock.

FF—Field on Falling Edge of the Clock

0 = The value of the FIELD signal will be 0 after the falling edge of the clock.
1 = The value of the FIELD signal will be 1 after the falling edge of the clock.

BR—Blanking on Rising Edge of the Clock

0 = The value of the $\overline{\text{BLANK}}$ signal will be 0 after the rising edge of the clock.
1 = The value of the $\overline{\text{BLANK}}$ signal will be 1 after the rising edge of the clock.

BF—Blanking on Falling Edge of the Clock

0 = The value of the $\overline{\text{BLANK}}$ signal will be 0 after the falling edge of the clock.
1 = The value of the $\overline{\text{BLANK}}$ signal will be 1 after the falling edge of the clock.

Note: All pins are general-purpose and can be programmed according to your requirements. The signal pin value only changes at the first cycle in which the line is valid. If a line is valid for more than one clock (CNT > 1), the signal holds its last assigned value.

Bits 8–13 and 17–18—Reserved

These bits are reserved and should be set to 0.

VDS—Video Data Select

This field selects the source of the video data or holds the last value of the data for the next CNT cycle.

00 = Select active video from display frame buffer (FIFO output).
01 = Select inactive (background) video from the background color buffer.
10 = Hold last value of data.
11 = Reserved.

INT—Interrupt

0 = Do not generate a interrupt to the core.
1 = Generate a maskable interrupt (end of frame) after this entry completes and before the next one begins.

LCYC/CNT—Loop Cycles/Clock Count

This field is a special field that is used for two purposes. It is used as the LCYC field if the LP bit is set to mark the beginning of a loop and it is used as a CNT field to count the number of clocks to hold a line. CNT is assumed as 1 if the LCYC field is valid (LP = 1 and at beginning of loop).

If LP = 0, then
CNT = the number of video clocks for this entry.

If LP = 1 and this is the beginning of the loop, then
LCYC = the number of loop cycles and CNT is already defined equal to 1.

If LP = 1 and this is the end of the loop, then
CNT = the number of video clocks for this entry.

Note: The value of the LCYC/CNT field should be ≥1 or erroneous operation will occur.

LP—Loop

This bit marks the beginning and end of a loop. The entry that marks the start of a loop should have the LP bit and the LCYC field set to the number of desired iterations. The entry that marks the end of a loop should have the LP bit set as well. Since only one bit marks the beginning and end of a loop, nested loops are not possible.

LST—LAST

0 = This is not the last valid entry.
1 = This is the last valid entry.

19.5 PROGRAMMING EXAMPLES

The following examples demonstrate how to program the video controller to support an NTSC or PAL interlaced display using Analog Devices ADV7176 video encoder with a few assumptions.

- The CCIR 601 4:2:2 video data format is used.
- The ADV7176 is controlled by the $\overline{\text{HSYNC}}$, $\overline{\text{BLANK}}$, and FIELD signals (MODE1:slave option) and has already been configured using I^2C.
- The image data resides in consecutive addresses of the memory.

19.5.1 NTSC Example

The NTSC (National Television Standards Committee) broadcast method defines a virtual screen of 525 lines where each line is 858 pixels long, but you can only see 485 of them. Each horizontal line contains 720 pixels during the active line period and the rest of the line's pixels are presented during the digital blanking. Figure 19-5 illustrates how the frame is divided into the following seven sections that correlate with the behavior of the FIELD and $\overline{\text{BLANK}}$ signals:

- Lines 1-3: frame 1 retrace blanking
- Lines 4-21: frame 0 blanking
- Lines 22-261: frame 0 active video for odd lines
- Lines 262-265: frame 0 retrace blanking
- Lines 266-284: frame 1 blanking
- Lines 285-524: frame 1 active video for even lines
- Line 525: frame 1 retrace blanking

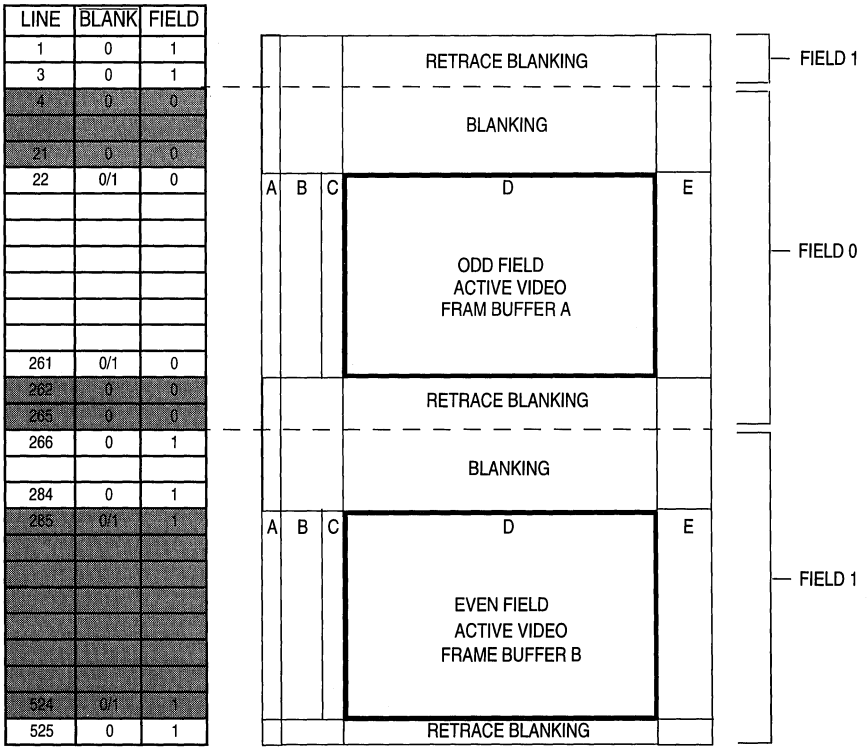


Figure 19-5. Interlaced NTSC Format

Figure 19-6 illustrates the horizontal timing of a single horizontal line, which is represented by five RAM entries:

- A—The section of the line where both blanking and $\overline{\text{HSYNC}}$ are asserted.
- B—The section of the line where $\overline{\text{HSYNC}}$ is negated and blanking is asserted.
- C—The section of the line where both signals are negated while the driven data is background
- D—The section of the line where both signals are negated while the driven data is the image data.
- E—The section of the line where blanking is asserted and $\overline{\text{HSYNC}}$ is negated.

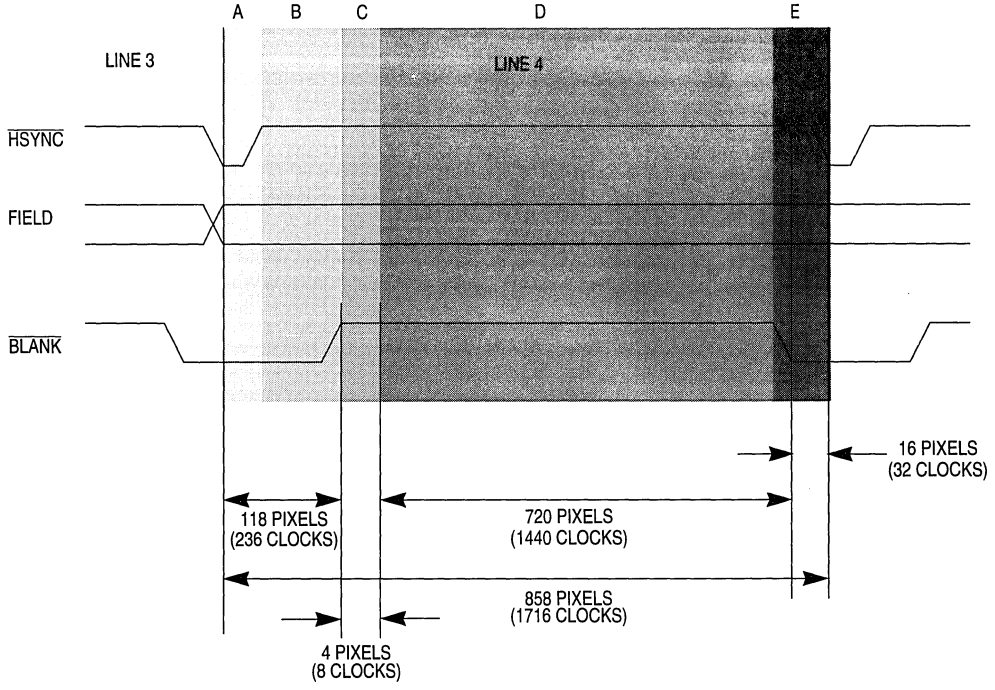


Figure 19-6. NTSC Horizontal Timing

19.5.1.1 NTSC PROGRAMMING PROCEDURE EXAMPLE. Use the following procedure to program your video controller using an ADV7176 video encoder in slave mode with video data in CCIR 4:2:2 format. A clock crystal provides 27MHz to CLK. Our website has a comprehensive example that includes using I^2C to program the ADV7176.

- Program the VBCB register with the components of the background video. For a black background, write 0x80108010 to the VBCB register.
- Program 0x07805A5A to the VFCR1. It defines a field of 240 lines and each line consists of 90 bursts of data. There is a GAP1 of one line (90 bursts) long between two consecutive lines due to interlace mode.
- Program the start address of the odd field to VFAA1 register.
- Program the start address of the even field to VFBA1 register. This address should be equal to VFAA1 address+0x5A0.
- Use Table 19-1 to configure the video controller RAM array.
- Write 0x02 to the VCMR to select RAM_1 and FIFO register set 1 as the active set.
- Reset the ADV7176 and program the default NTSC settings, except for mode register 1 (0x02), timing register 0 (0x02), and mode register 2 (0x08).
- Convert your data to CbYCrY format for the ADV7176. Assuming each color ranges between 0 and 100, use the following equations to convert from RGB to YC_rC_b for the ADV7176:

```
void SetPixelRGB (int col, int row, PALLETE color)
{
    int location;
    VYUY *address;
    location = screen.burstlength * (row+25) + ((col+55) / 2*4);
    address = (VYUY *) (location + (int) screen.base);
    if (col % 2)
    {
        address -> Y1 = 209 * (color.red + color.green + color.blue) / 300 +16
        address -> V = color.blue - color.red/4 - color.green*3/4 = 128;
    }
    else
    {
        address -> Y2 = 209 * (color.red + color.green + color.blue) / 300 + 16
        address -> U = color.red - color.green*3/4 - color.blue/4 + 128;
    }
}
```

- Program 0x2043 to the VCCR to operate the video controller.

Table 19-1. Video RAM Array Loaded with NTSC Example

RAM ENTRY	RAM WORD FIELD									LINE DESCRIPTION
	Hx	Vx	Fx	Bx	VDS	INT	LCYC	LP	LST	
0	00	00	11	00	01	0	3	1	0	Entry 0: LP=1—Beginning of loop. LCYC=3—Loop entries 0-3 three times. CNT=1—Hold this entry for one video clock.
1	11	00	11	00	01	0	243	0	0	Entry 1: CNT=243—Hold this entry for 243 video clocks.
2	11	00	11	00	01	0	1440	0	0	Entry 2: CNT=1440—Hold this entry for 1440 video clocks.
3	11	00	11	00	01	0	32	1	0	Entry 3: LP=1—End of loop. CNT=32—Hold this entry for 32 video clocks.
4	00	00	00	00	01	0	18	1	0	Lines 4-21, FIELD = 0, $\overline{\text{BLANK}}$ is asserted (vertical blanking)
5	11	00	00	00	01	0	243	0	0	
6	11	00	00	00	01	0	1440	0	0	
7	11	00	00	00	01	0	32	1	0	
8	00	00	00	00	01	0	240	1	0	Lines 22-261. ODD field active area, the horizontal line is repeated 240 times
9	11	00	00	00	01	0	235	0	0	
10	11	00	00	11	01	0	8	0	0	
11	11	00	00	11	00	0	1440	0	0	
12	11	00	00	00	01	0	32	1	0	
13	00	00	00	00	01	0	4	1	0	Lines 262-265, FIELD = 0, $\overline{\text{BLANK}}$ asserted (vertical; blanking)
14	11	00	00	00	01	0	243	0	0	
15	11	00	00	00	01	0	1440	0	0	
16	11	00	00	00	01	0	32	1	0	
17	00	00	11	00	01	0	19	1	0	Lines 266-284, FIELD = 1, $\overline{\text{BLANK}}$ asserted (vertical; blanking)
18	11	00	11	00	01	0	243	0	0	
19	11	00	11	00	01	0	1440	0	0	
20	11	00	11	00	01	0	32	1	0	
21	00	00	11	00	01	0	240	1	0	Lines 285-524. EVEN Field active area
22	11	00	11	00	01	0	235	0	0	
23	11	00	11	11	01	0	8	0	0	
24	11	00	11	11	00	0	1440	0	0	
25	11	00	11	00	01	0	32	1	0	
26	00	00	11	00	01	0	1	1	0	Line 525, FIELD = 1, $\overline{\text{BLANK}}$ is asserted (vertical blanking)
27	11	00	11	00	01	0	243	0	0	
28	11	00	11	00	01	0	1440	0	0	
29	11	00	11	00	01	1	32	1	1	

NOTE: HX = HR:HF and VX = VR:VF.

19.5.2 PAL Example

The PAL (Phase Alternation Line) broadcast method defines a virtual screen of 625 lines, which are 864 pixels long. However, you can only see 576 of these lines. Each horizontal line contains 720 pixel during the active line period (which is visible to you) with the rest of the line's pixels presented during digital blanking. Figure 19-7 illustrates how the frame is divided into six sections that correlate with the behavior of the FIELD and $\overline{\text{BLANK}}$ signals:

- Lines 1-22: Frame 0 blanking
- Lines 23-310: Frame 0 active video for odd lines
- Lines 311-312: Frame 0 for retrace blanking
- Lines 313-335: Frame 1 blanking
- Lines 336-623: Frame 1 active video for even lines
- Lines 624-625: Frame 1 retrace blanking

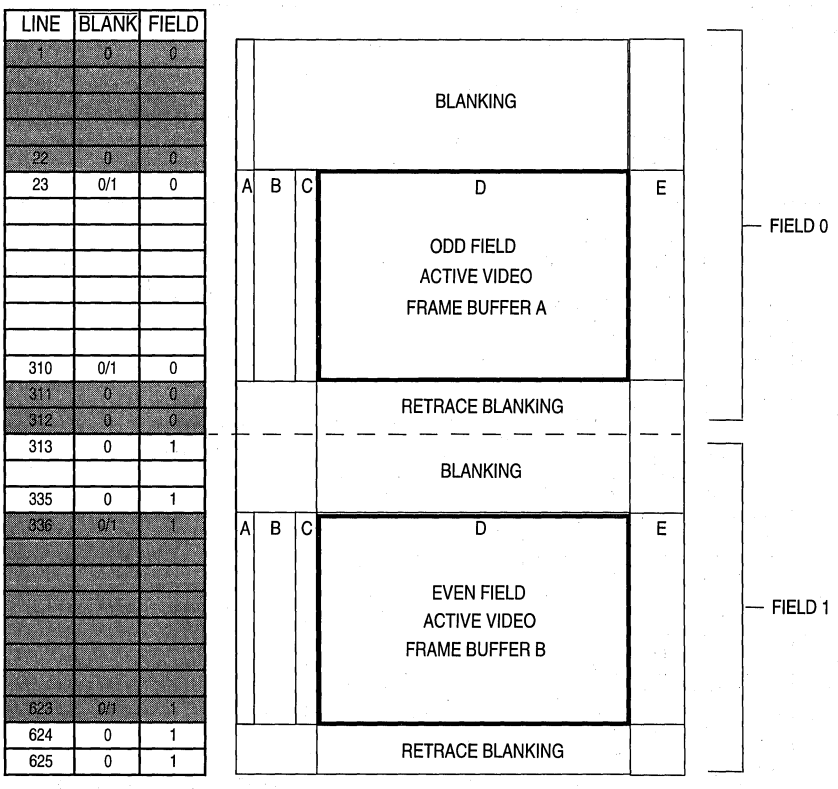


Figure 19-7. Interlaced PAL Format

Figure 19-8 illustrates the horizontal timing of a single horizontal line that is represented by five RAM entries:

- A defines the section of the line where the both the $\overline{\text{BLANK}}$ and $\overline{\text{HSYNC}}$ signals are asserted.
- B defines the section of the line where the $\overline{\text{HSYNC}}$ signal is negated and $\overline{\text{BLANK}}$ is asserted.
- C defines the section of the line where the $\overline{\text{HSYNC}}$ and $\overline{\text{BLANK}}$ signals are negated while the driven data is background.
- D defines the section of the line where the $\overline{\text{HSYNC}}$ and $\overline{\text{BLANK}}$ signals are negated while the driven data is the image data.
- E defines the section of the line where the $\overline{\text{BLANK}}$ signal is asserted and $\overline{\text{HSYNC}}$ is negated.

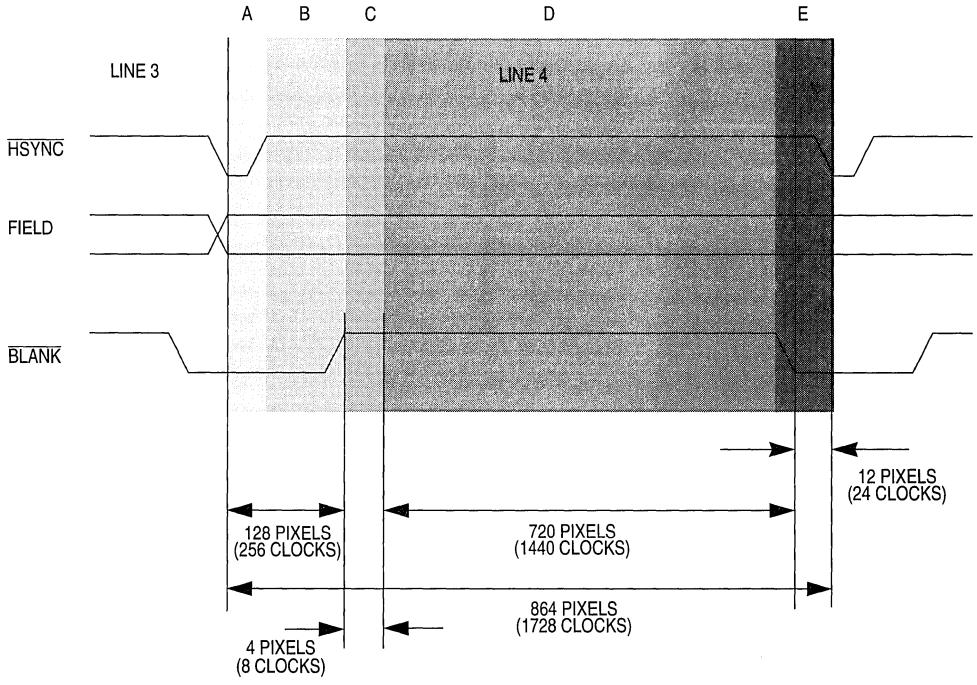


Figure 19-8. PAL Horizontal Timing

19.5.2.1 PAL PROGRAMMING PROCEDURE EXAMPLE. Use the following procedure to program your video controller using an ADV7176 video encoder in slave mode with video data in CCIR 4:2:2 format.

- Program the VBCB register with the components of the background video. For a black background write 0x80108010 to the VBCB register.
- Program 0x09005A5A to the VFCR1. It defines a field of 288 lines, in which each line consists of 90 burst of data. There is a GAP1 of one line (90 bursts) long between two consecutive lines due to interlace mode.
- Program the start address of the odd field to VFAA1 register.
- Program the start address of the even field to the VFBA1 register. This address should be equal to the VFAA1 address+0x5A0.
- Use Table 19-2 to configure the video controller RAM array.
- Write 0x02 to the VCMR to select RAM_1 and FIFO register set 1 as the active set.
- Program 0x2041 to the VCCR to operate the video controller.

Table 19-2. Video RAM Word Loaded with PAL Example

RAM ENTRY	RAM WORD FIELD									LINE DESCRIPTION
	Hx	Vx	Fx	Bx	VDS	INT	LCYC	LP	LST	
0	00	00	00	00	01	0	22	1	0	Lines 1-22, FIELD = 0, $\overline{\text{BLANK}}$ is asserted (vertical blanking)
1	11	00	00	00	01	0	263	0	0	
2	11	00	00	00	01	0	1440	0	0	
3	11	00	00	00	01	0	24	1	0	
4	00	00	00	00	01	0	288	1	0	Entry 4: Beginning of loop(loop entries 4-8 288 times). Assert HSYNC and $\overline{\text{BLANK}}$ for 1 video clock.
5	11	00	00	00	01	0	255	0	0	Entry 5: Assert $\overline{\text{BLANK}}$ and negate $\overline{\text{HSYNC}}$ for 255 video clocks.
6	11	00	00	11	01	0	8	0	0	Entry 6: Negate $\overline{\text{BLANK}}$ for 8 video clocks. Display video data from the VBCB.
7	11	00	00	11	00	0	1440	0	0	Entry 7: Display active video data from frame buffer for 1440 video clocks (720 pixels).
8	11	00	00	00	01	0	24	1	0	Entry 8: Assert $\overline{\text{BLANK}}$ for 24 video clocks.
9	00	00	00	00	01	0	2	1	0	Lines 311-312, FIELD = 0, $\overline{\text{BLANK}}$ asserted (vertical; blanking)
10	11	00	00	00	01	0	263	0	0	
11	11	00	00	00	01	0	1440	0	0	
12	11	00	00	00	01	0	24	1	0	
13	00	00	11	00	01	0	23	1	0	Lines 313-335, FIELD = 1, $\overline{\text{BLANK}}$ asserted (vertical; blanking)
14	11	00	11	00	01	0	263	0	0	
15	11	00	11	00	01	0	1440	0	0	
16	11	00	11	00	01	0	24	1	0	
17	00	00	11	00	01	0	288	1	0	Lines 336-623. EVEN field active area
18	11	00	11	00	01	0	255	0	0	
19	11	00	11	11	01	0	8	0	0	
20	11	00	11	11	00	0	1440	0	0	
21	11	00	11	00	01	0	24	1	0	
22	00	00	11	00	01	0	2	1	0	Lines 624-625, FIELD = 1, $\overline{\text{BLANK}}$ is asserted (vertical blanking)
23	11	00	11	00	01	0	263	0	0	
24	11	00	11	00	01	0	1440	0	0	
25	11	00	11	00	01	1	24	1	1	

NOTE: HX = HR:HF and VX = VR:VF.

SECTION 20
DEVELOPMENT CAPABILITIES AND INTERFACE

This section discusses the on-chip features that are used during the development phase. Background debug monitors and emulators are used to interface with this MPC823 capability. Emulators require a level of control and observation that are in sharp contrast to the trend of modern microcomputers and microprocessors in which many bus cycles are directed to internal resources and are not externally visible. The same is true for bus analyzers. To enhance support for development tools, some of the development support functions are implemented in the silicon. Program flow tracking, watchpoint and breakpoint generation, and emulation systems that control core activity are just some of the features that allow you to efficiently debug systems based on the MPC823.

20.1 FEATURES

The following list summarizes the development features of the MPC823:

- Program Flow Tracking
  - ❑ Instruction show cycles
  - ❑ Data show cycles
  - ❑ Branching
  - ❑ Exception trap
- Watchpoint and Breakpoint Generation
  - ❑ Four hardware breakpoints
  - ❑ Five watchpoint sources
- Simple Hardware Interface
  - ❑ High-speed data interface
  - ❑ Internal status pins
  - ❑ Freeze indication
- Rich Control Register Set

20.2 PROGRAM FLOW TRACKING

The MPC823 provides many options for tracking program flows that impact performance in varying degrees. The information provided while tracking code flow can be compressed and captured externally and then parsed by a post-processing program using the microarchitecture defined here. The program instruction flow is visible on the external bus when the MPC823 is programmed to operate in serialized mode and show all fetch cycles on the external bus. When working in this mode, although tracking of the program instruction flow is simpler, the performance of the MPC823 is much lower than when working in regular mode. See **Section 20.6.2.5 Instruction Support Control Register** for more details about programming the core to operate in this mode.

The MPC823 implements a prefetch queue combined with parallel, out-of-order, and pipelined execution. These features, plus the fact that most fetch cycles are performed internally from the instruction cache, increases the performance but makes it very difficult to provide you with the real program trace. Instructions progress inside the core from fetch to retirement. An instruction retires from the machine only after it and all preceding instructions finish execution with no exception. Therefore, only retired instructions can be considered *a*rchitecturally executed.

Reporting program trace during retirement significantly complicates visibility and increases the die size for the two reasons—more than one instruction can retire in a clock cycle and it is harder to report on indirect branches during retirement. Because of this, program trace is reported during fetch and helps to reconstruct the instructions that actually retire after fetch canceled instructions are reported. Instructions are fetched sequentially until branches (direct or indirect), exceptions, or interrupts appear in the program flow or until a stall in execution forces the machine to avoid fetching the next address. These instructions can be architecturally executed or they can be canceled in any stage of the machine pipeline.

To reconstruct a program trace, you need the program code in addition to the following MPC823 information:

- A description of the last fetched instruction (stall, sequential, branch not taken, branch direct taken, branch indirect taken, interrupt/exception taken).
- The addresses of all indirect flow changes targets. Indirect flow changes include all branches using the link and count registers as the target address, all interrupts/ exceptions, as well as **rfi** and **mtmsr** because it may cause a context switch.
- The number of instructions canceled on each clock.

20.2.1 Basic Operation

20.2.1.1 THE INTERNAL HARDWARE. To make the events that occur in the machine visible, a few dedicated pins are used. Also, a special bus cycle attribute called program trace cycle is defined. The program trace cycle attribute is attached to all fetch cycles that result from indirect flow changes. When program trace recording is required, you must program the appropriate registers to ensure that these cycles are visible on the external bus.

The internal visible sync (VSYNC) signal, when asserted, forces all fetch cycles marked with the program trace cycle attribute to be visible on the external bus, even if their data is found in one of the internal devices. To enable the external hardware to properly synchronize with the internal activity of the core, VSYNC assertion and negation forces the machine to synchronize and marks the first fetch after the synchronization as a program trace cycle that be seen on the external bus. For more information about the activity of the external hardware during program trace, refer to **Section 20.2.1.2 The External Hardware**

Note: To keep the pin count of the chip as low as possible, the VSYNC signal is not implemented as one of the chip's external pins. Instead, it is asserted and negatedT using the serial interface implemented in the development port. For more information on this interface, refer to **Section 20.4.3 The Development Interface Port**. Forcing the core to show all fetch cycles marked with the program trace cycle attribute can be accomplished by either asserting the VSYNC signal or by programming the ISCT_SER fieldin the instruction support control (ICTRL) register. For more information, see **Section 20.2.2 Controlling Instruction Fetch Show Cycles**.

When the VSYNC signal is asserted, all fetch cycles marked with the program trace cycle attribute become visible on the external bus. These cycles generate regular bus cycles when the instructions reside in one of the external devices or generate address-only cycles when the instructions are in one of the internal devices. When the VSYNC signal is asserted, some performance degradation occurs because of the additional external bus cycles. Since this performance degradation is expected to be very small, you can program the machine to show all indirect flow changes, perform these additional external bus cycles, and maintain the same behavior when the VSYNC signal is asserted and negated. For more information, see **Section 20.6.2.5 Instruction Support Control Register**.

The status pins are divided into two groups—the instruction queue status and the history buffer flush status.

- VF[0:2]—Visible Instruction Queue Flushes Status

  Instruction queue status denotes the type of the last fetched instruction or how many instructions were flushed from the instruction queue. These status pins are used for both functions because queue flushes only happen in clocks where there is no fetch type information to be reported, as shown in Table 20-1.

  000 = None.
  001 = 1 instruction was flushed from the instruction queue.
  010 = 2 instructions were flushed from the instruction queue.
  011 = 3 instructions were flushed from the instruction queue.
  100 = 4 instructions were flushed from the instruction queue.
  101 = 5 instructions were flushed from the instruction queue.
  110 = Reserved.
  111 = Special case. See **Section 20.2.1.1 The Internal Hardware**.

Table 20-1. VF Instruction Type Encoding

VF	INSTRUCTION TYPE	VF NEXT CLOCK WILL HOLD
000	None	More instruction type information
001	Sequential	
010	Branch (direct or indirect) not taken	
011	VSYNC was asserted/negated and therefore the next instruction will be marked with the program trace cycle attribute	
100	Interrupt/exception taken, the target will be marked with the program trace cycle attribute	Queue flush information[2]
101	Branch indirect taken, **rfi**, **mtmsr**, **isync** and in some cases **mtspr**, the target will be marked with the program trace cycle attribute[1]	
110	Branch direct taken	
111	Branch (direct or indirect) not taken	

NOTES: Unless the next clock VF = 111, refer to **Section 20.2.1.1 The Internal Hardware**.

- VFLS[0:1]—Visible History Buffer Flushes Status

  History buffer flushes status indicates the number of instructions that are flushed from the history buffer on this clock.

  00 = None.
  01 = One instruction was flushed from the history buffer.
  10 = Two instructions were flushed from the history buffer.
  11 = Used for debug mode indication and should be ignored by the program trace external hardware. For details, refer to **Section 20.4.2 Debug Mode**.

20.2.1.1.1 Special Case Queue Flush Information. There is one special case where the queue flush information is expected on the VF pins. This is easily monitored since the only case where this can happen is when VF =111 and the maximum number of possible queue flushes is five.

20.2.1.1.2 Program Trace In Debug Mode. When entering debug mode an interrupt/exception is reported on the VF pins (VF=100) and a cycle marked with the program trace cycle is externally visible. When the CPU is in debug mode, the VF pins equal 000 and the VFLS pins equal 11. For more information on the MPC823 debug mode, refer to **Section 20.4 Hardware Development System Interface**.

If the VSYNC signal is asserted/negated while the core is in debug mode, this information is announced when the first VF pins report as the core returns to regular mode. If VSYNC was not changed while in debug mode, the first VF pins report will be encoded as VF=101 (indirect branch) due to the **rfi** instruction being issued. In both cases, the first instruction fetch after debug mode is marked with the program trace cycle attribute and is externally visible. When the MPC823 external bus is configures to operate at half the speed of the internal system (EBDF=1), the VF and VFLS pins will not report fetch and flush information for the program trace capability. However, the internal freeze state of the processor will be reported on the VFLS pins.

20.2.1.1.3 Sequential Instructions Marked As Indirect Branch. There are instances where nonbranch or sequential instructions affect the machine similar to the way that indirect branch instructions affect it. These sequential instructions include **rfi**, **mtmsr**, **isync**, and **mtspr** to the BAR, CMPA-CMPH, COUNTA, COUNTB, ICTRL, ICR, LCTRL1, LCTRL2, and DER registers.

The core marks these instructions as indirect branch instructions (VF = 101) and the following instruction address is marked with the program trace cycle attribute, as if it was an indirect branch target. Therefore, when one of these special instructions is detected in the core, the address of the following instruction is externally visible. The reconstructing software is now able to correctly evaluate the effect of these instructions.

20.2.1.2 THE EXTERNAL HARDWARE. When a program trace is needed, the external hardware must sample the status pins—VF and VFLS—of every clock and mark the address of all cycles with the program trace cycle attribute. Program trace is used in various ways, but back trace and window trace are the most common methods.

20.2.1.2.1 Back Trace. This is useful when a record of the program trace is needed before an event occurs, such as system failure. If back trace is needed, the external hardware should start sampling VF and VFLS pins and the addresses of all cycles marked with the program trace cycle attribute immediately after reset is negated. Since the instruction show cycles programming defaults to show all out of reset, all cycles marked with the program trace cycle attribute are visible on the external bus. VSYNC should be asserted sometime after reset and negated when the actual event occurs. If show all is not the preferred mode for the instruction show cycles before the event actually occurs, VSYNC must be asserted before exiting show all mode. If the timing of the event in question is unknown, it is possible to use cyclic buffers. After the VSYNC signal is negated, the trace buffer contains the program flow trace of the program executed before the event in question occurred.

20.2.1.2.2 Window Trace. This is useful when a record of the program trace between two events is required. The VSYNC pin should be asserted between these two events. After VSYNC is negated, the trace buffer will contain information describing the program trace of the program executed between the two events.

20.2.1.2.3 Synchronizing the Trace Window to the Internal Core Events. The VSYNC signal is asserted or negated using the serial interface implemented in the development port. To synchronize the assertion or negation to an internal core event, the internal breakpoint hardware should be used with debug mode. This method is available only when debug mode is enabled. For more information on debug mode, refer to **Section 20.4 Hardware Development System Interface**.

To synchronize the trace window to the internal core events, follow these steps:

1. Enter debug mode either straight from reset or when using a debug mode request.
2. Program the hardware to break on the event that marks the start of the trace window using the control registers defined in **Section 20.3 Generating Watchpoints And Breakpoints**.
3. Enable debug mode entry for the programmed breakpoint in the debug enable register. See **Section 20.6.3.2 Debug Enable Register** for details.
4. Return to the regular code run. The hardware generates a breakpoint when the event in question is detected and the machine enters debug mode.
5. Program the hardware to break on the event that marks the end of the trace window.
6. Assert the VSYNC signal.
7. Return to the regular code run. The first report on the VF pins is VSYNC, where VF = 011. The external hardware starts sampling the program trace information after the VF pins indicate VSYNC. The hardware generates a breakpoint when the event in question is detected and the machine enters debug mode.
8. Negate the VSYNC signal.
9. Return to the regular code run and issue an **rfi** instruction. The first encoding on the VF pins is VSYNC, where VF = 011. The external hardware stops sampling the program trace information after recognizing VSYNC on the VF pins.

20.2.1.2.4 Detecting the Trace Window Start Address. When using back trace, latching VF, VFLS, and the address of the cycles marked program trace cycle should all start immediately after reset is negated. The start address is the first address in the program trace cycle buffer. When using window trace, latching of VF, VFLS, and the address of the cycles marked as program trace cycle should all start immediately after the first VSYNC is recognized on the VF pins. The start address of the trace window should be calculated according to the first two VF pin reports. Assume VF1 and VF2 are the first two VF pin reports and T1 and T2 are the two addresses of the first two cycles marked with the program trace cycle attribute that were latched in the trace buffer. Use the following table to calculate the trace window start address.

Table 20-2. Detecting the Trace Buffer Starting Point

VF1	VF2	STARTING POINT	DESCRIPTION
011 VSYNC	001 Sequential	T1	VSYNC is asserted and followed by a sequential instruction. the start address is T1.
011 VSYNC	110 Branch Direct Taken	T1 - 4 + Offset(T1 - 4)	VSYNC is asserted and followed by a taken direct branch. the start address is the target of the direct branch.
011 VSYNC	101 Branch Indirect Taken	T2	VSYNC is asserted and followed by a taken indirect branch. the start address is the target of the indirect branch.

20.2.1.2.5 Detecting VSYNC Assertion/Negation. Because the VF pins are used to report both instruction type and queue flush information, the external hardware must take special care when trying to detect the assertion/negation of VSYNC. When VF = 011, it is a VSYNC assertion/negation report only if the prior value of VF was 000, 001, or 010.

20.2.1.2.6 Detecting the Trace Window End Address. The information on the status pins that describes the last fetched instruction and last queue/history buffer flush changes every clock. Cycles marked as program trace cycle are generated on the external bus only when the system interface unit arbitrates over the external bus. Therefore, there is a delay between the report that a cycle marked as program trace cycle is performed and the actual time that this cycle can be detected on the external bus.

When you negate VSYNC using the serial interface of the development port, the core delays reporting that VSYNC occurred on the VF pins until all addresses marked with the program trace cycle attribute are externally visible. Therefore, the external hardware should stop sampling VF, VFLS, and the address of the cycles marked as program trace cycle immediately after VF = VSYNC. The last two instructions reported on the VF pins are not always valid. Therefore, at the last stage of the reconstruction software, the last two instructions should be ignored.

20.2.1.3 COMPRESSION OF CANCELLED INSTRUCTIONS. To store all the information generated on the pins during program trace (5 bits per clock + 30 bits per show cycle), a large memory buffer is required. However, since this information includes events that were cancelled, compression is possible and can be very beneficial in this situation. External hardware can be added to eliminate all canceled instructions and reports only on taken or not taken branches, indirect flow change, and the number of sequential instructions after the last flow change.

20.2.2 Controlling Instruction Fetch Show Cycles

Instruction fetch show cycles are controlled by the bits in the ICTRL register and the state of the VSYNC signal. The following table defines the level of fetch show cycles generated by the core. Table 20-3 shows the types of fetch show cycles determined by the ISCT_SER field. A cycle marked with the program trace cycle attribute is generated for any change in the state of VSYNC.

Table 20-3. Fetch Show Cycle Types

VSYNC	ISCT_SER INSTRUCTION FETCH SHOW CYCLE CONTROL FIELD	SHOW CYCLES GENERATED
X	000	All Fetch Cycles
X	X01	All Change of Flow (Direct and Indirect)
X	X10	All Indirect Change of Flow
0	X11	No Show Cycles Are Performed
1	X11	All Indirect Change of Flow

NOTE: Only cycles that access storage assert the $\overline{TS}$ signal. All cycles that involve "show cycles" are marked by asserting the $\overline{STS}$ signal. When you need to sample the show cycle address and attributes, the $\overline{STS}$ signal should be enabled by programming the DBGC field of the SIUMCR. See **Section 12.12.1.1 SIU Module Configuration Register** for details.

20.3 GENERATING WATCHPOINTS AND BREAKPOINTS

When detected, watchpoints are reported to the external world on dedicated pins but do not change the timing and flow of the machine. When breakpoints are detected, they force the machine to branch to the appropriate exception handler. The core supports watchpoints that are generated inside the core as well as breakpoints that are generated inside and outside the core.

In the core, as in other RISC processors, saving and restoring the machine state on the stack during exception handling is done in the software. When the software is in the middle of saving and restoring, the MSR_{RI} bit is cleared. Exceptions that occur are handled by the core when the MSR_{RI} bit is clear and they result in a nonrestartable machine state. For more information refer to **Section 6.3.4.1 Restartability After An Interrupt**.

In general, breakpoints are recognized in the core only when the MSR_{RI} bit is set, which guarantees machine restartability after a breakpoint. In this working mode, breakpoints are masked. There are times when it is preferable to enable breakpoints even when the MSR_{RI} bit is clear, even though there is a risk of causing a nonrestartable machine state. In programmable nonmasked mode, an external development system can choose to assert a nonmaskable external breakpoint. Watchpoints are not masked and are always reported on the external pins, regardless of the value of the MSR_{RI} bit. The counters, although they are counting watchpoints, are part of the internal breakpoints logic and are not decremented when the core is in masked mode and the MSR_{RI} bit is clear.

Internal watchpoints are generated when a user-programmable set of conditions are met. Internal breakpoints can be programmed to be generated either when one of the internal watchpoints is asserted or after an internal watchpoint is asserted for user-programmable times. Programming a certain internal watchpoint to generate an internal breakpoint can be done either in the software, by setting the corresponding software trap enable bit, or on-the-fly using the serial interface of the development port to set the corresponding trap enable bit.

External breakpoints can be generated by any of the system peripherals, including those found on or outside the MPC823 or those found by an external development system. Peripherals on the external bus use the serial interface of the development port to assert an external breakpoint.

20.3.1 Internal Watchpoints and Breakpoints

Internal watchpoints and breakpoints are used in software debugging and the sources are illustrated in Figure 20-1. For the recoverable interrupt bit of the MSR, see **Section 6 The PowerPC Core**. Watchpoints do not stop your code from executing, but they indicate when you have passed a certain testing point. Breakpoints are actually the points at which execution is stopped. For more information on external breakpoint support, refer to **Section 20.4 Hardware Development System Interface**. Internal breakpoint and watchpoint support is based on:

- Eight comparators that compare information on instruction and load/store cycles
- Two counters
- Two AND-OR logic structures

The comparators perform a comparison on the instruction address (I-address), the load/store address (L-address), and the load/store data (L-data). The comparators can detect the following conditions:

- Equal to
- Not equal to
- Greater than
- Less than

Greater-than-or-equal-to and less-than-or-equal-to are easily obtained from these four conditions. Refer to **Section 20.3.1.5 Generating Compare Types** for more information.

Using the AND-OR logic structures "in range" and "out of range", detections of address and data comparators are supported. Using the counters, you can program a breakpoint to be recognized after an event has been detected after a predefined number of times.

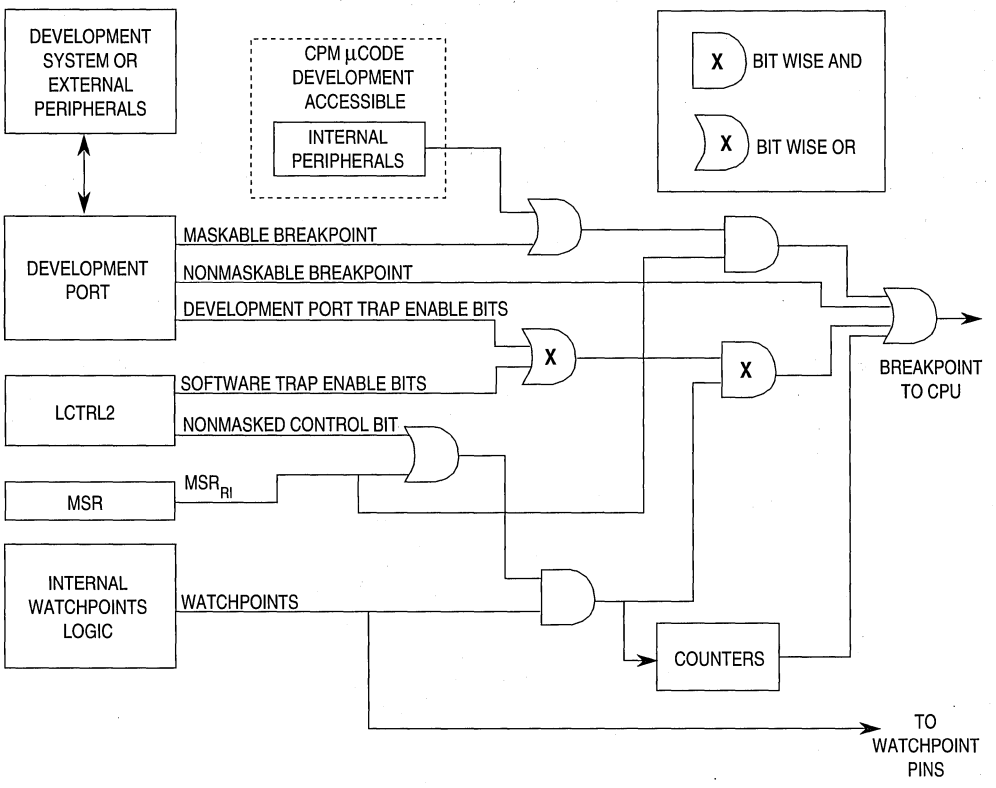


Figure 20-1. Watchpoint and Breakpoint Support in the Core

The L-data comparators operate on load or store fixed-point data. When operating on fixed-point data the L-data comparators perform a comparison on bytes, half-words, and words. They treat numbers as either signed or unsigned values. The comparators generate match events and then instruction match events enter the instruction AND-OR logic where the instruction watchpoints and breakpoint are generated. The asserted instruction watchpoints can generate the instruction breakpoint. Two different events can decrement one of the counters. When a counter on one of the instruction watchpoints expires, the instruction breakpoint is asserted.

The instruction watchpoints and load/store match events on the address/data comparators enter the load/store AND-OR logic where the load/store watchpoints and breakpoint are generated. When aserted, the load/store watchpoints can generate the load/store breakpoint or decrement one of the counters. When a counter on one of the load/store watchpoints expires, the load/store breakpoint is asserted.

Watchpoints progress in the machine and are reported when they retire. Internal breakpoints progress in the machine until they reach the top of the history buffer when the machine branches to the breakpoint exception routine. So the breakpoint features can be used without restricting the software, the address of the load/store cycle that generated the load/store breakpoint is not stored in the data address register (DAR). In a load/store breakpoint, the address of the load/store cycle that generated the breakpoint is stored in the breakpoint address register (BAR). There are many types of internal watchpoints and breakpoints:

- Four I-Address Comparators Supporting Equal, Not Equal, Greater Than, and Less Than.
- Two L-Address Comparators Supporting Equal, Not Equal, Greater Than, and Less Than.
- Two L-Data Comparators Supporting Equal, Not Equal, Greater Than, and Less Than.
- No Internal Breakpoint or Watchpoint Support for Unaligned Words and Half-Words.
- The L-Data Comparators Can Be Programmed to Treat Fixed-Point Numbers as Signed or Unsigned Values.
- Combined Comparator Pairs to Detect In and Out of Range Conditions, Including Either Signed or Unsigned Values On the L-Data.
- A Programmable AND-OR Logic Structure Between the Four Instruction Comparators Results in Five Outputs, Four Instruction Watchpoints, and One Instruction Breakpoint.
- A Programmable AND-OR Logic Structure Between the Four Instruction Watchpoints and the Four Load/Store Comparators Results in Three Outputs, Two Load/Store Watchpoints, and One Load/Store Breakpoint.
- Five Watchpoint Pins, Three For the Instruction and Two For the Load/Store.
- Two Dedicated 16-Bit Down Counters. Each Can Be Programmed to Count Either an Instruction Watchpoint or a Load/Store Watchpoint. Only Architecturally Executed Events are Counted.
- On-The-Fly Trap Enable Programming of the Different Internal Breakpoints Using the Serial Interface of the Development Port. Software Control Is Also Available.
- Watchpoints Do Not Change the Timing of the Machine.
- Internal Breakpoints and Watchpoints Are Detected on the Instruction During Instruction Fetch.
- Internal Breakpoints and Watchpoints Are Detected on the Load/Store During Load/Store Bus Cycles.
- Instruction and Load/Store Breakpoints and Watchpoints Are Handled on Retirement and Then Reported.

- Breakpoints and Watchpoints on Recovered Instructions (As a Result of Exceptions, Interrupts, or Miss Prediction) Are Not Reported and Do Not Change the Timing of the Machine.
- Instructions with Instruction Breakpoints Are Not Executed. The Machine Branches to the Breakpoint Exception Routine Before it Executes the Instruction.
- Instructions with Load/Store Breakpoints Are Executed. The Machine Branches to the Breakpoint Exception Routine After it Executes the Instruction. The Address of the Access is Placed in the Breakpoint Address Register.
- Load/Store Multiple and String Instructions with Load/Store Breakpoints First Finish Execution and Then the Machine Branches to the Breakpoint Exception Routine.
- Load/Store Data Compare is Made On the Load/Store, After Swap in Store Accesses and Before Swap in Load Accesses (As the Data Appears on the Bus).
- Internal Breakpoints Operate with a Context-Dependent Filter.
- Both "go to x" and "continue" Working Modes are Supported for Instruction Breakpoints.

20.3.1.1 RESTRICTIONS. There are times when the same watchpoint can be detected more than once during the execution of a single instruction. For example, a load/store watchpoint detected on more than one transfer when executing load/store multiple/string or load/store watchpoint detected on more than one byte when working in byte mode. In these cases only one watchpoint of the same type is reported for a single instruction. Similarly, only one watchpoint of the same type can be counted in the counters for a single instruction. Since watchpoint events are reported when the instruction that caused the event retires (more than one instruction can retire from the machine in a single clock), ensuing events can be reported in the same clock. Moreover, if the same event is detected on more than one instruction (tight loops or range detection) can just be reported once. The internal counters count correctly in these cases.

20.3.1.2 BYTE AND HALF-WORD WORKING MODES. You can use watchpoints and breakpoints to detect matches on bytes and half-words when the byte/half-word is accessed in a load/store instruction of larger data widths. For example, when loading a table of bytes using a series of load word instructions.) To use this feature in word mode, you should write the required match value to the correct half-word of the data comparator and to the mask in the L-data comparator. If you prefer to break on bytes, the byte mask for each L-comparator and the bytes to be matched must be written in the data comparator.

Since bytes and half-words can be accessed using a larger data width instruction, it is impossible for you to predict the exact value of the L-address lines when the requested byte/ half-word is accessed. If the matched byte is byte 2 of the word and is accessed using a load word instruction, the L-address value will be of the word (byte 0). Therefore, the core masks the two least-significant bits of the L-address comparators whenever a word access is performed and the least-significant bits whenever a half-word access is performed. Address range is only supported when aligned according to the access size.

Byte Working Mode Example

Data size: Byte.
Address: 0x00000003.
Data value: Greater than 0x07 and less than 0x0c.

Programming options:
One L-address comparator = 0x00000003 and program for equal.
One L-data comparator = 0x00000007 and program for greater than.
One L-data comparator = 0x0000000c and program for less than.
CGBMSK and CHBMSK fields of the LCTRL1 = 0xe.
Both L-data comparators program to byte mode.

Result:
The event will be correctly detected, regardless of the load/store instruction the compiler chooses for this access.

Half-Word Working Mode Example 1

Data size: Half-word.
Address: Greater than 0x00000000 and less than 0x0000000c.
Data value: Greater than 0x4e204e20 and less than 0x9c409c40.

Programming option:
One L-address comparator = 0x00000000 and program for greater than.
One L-address comparator = 0x0000000c and program for less than.
One L-data comparator = 0x4e204e20 and program for greater than.
One L-data comparator = 0x9c409c40 and program for less than.
CGBMSK and CHBMSK fields of the LCTRL1 = 0xe.
Both L-data comparators program to half-word mode.

Result:
The event will be correctly detected as long as the compiler does not use a load/store instruction with data size of byte.

Half-Word Working Mode Example 2

Data size: Half-word.
Address: Greater than or equal to 0x00000002 and less than 0x0000000e.
Data value: Greater than 0x4e204e20 and less than 0x9c409c40.

Programming option:
One L-address comparator = 0x00000001 and program for greater than.
One L-address comparator = 0x0000000e and program for less than.
One L-data comparator = 0x4e204e20 and program for greater than.
One L-data comparator = 0x9c409c40 and program for less than.
CGBMSK and CHBMSK fields of the LCTRL1 = 0xe.
Both L-data comparators program to half-word or word mode.

Result:
The event will be correctly detected if the compiler chooses a load/store instruction with data size of half-word. If the compiler chooses load/store instructions with data size greater than half-word (word, multiple), there might be some false detections.

This example uses Figure 20-2 to show the possible false detects that should be ignored by the software that handles the breakpoints.

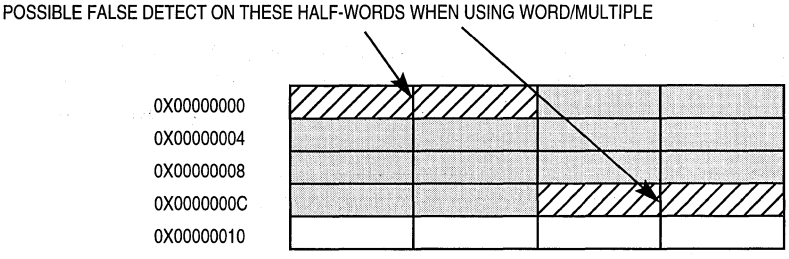


Figure 20-2. Example 2 False Detect on Watchpoint/Breakpoint

20.3.1.3 CONTEXT-DEPENDENT FILTER. The core can only be programmed to recognize internal breakpoints when the MSR_{RI} bit is set or to always recognize internal breakpoints. When it is programmed only to recognize internal breakpoints or when MSR_{RI} = 1, all parts of the code can be debugged, except when the save and restore register 0 (SRR0), save and restore register 1 (SRR1), data address register (DAR), and data storage interrupt status register (DSISR) are busy and MSR_{RI} = 0.

When the core is programmed to recognize internal breakpoints, it is possible to debug all parts of the code. However, if an internal breakpoint is recognized when MSR_{RI} = 0 (registers SRR0 and SRR1 are busy), the machine enters into a nonrestartable state. For more information refer to **Section 6.3.4.1 Restartability After An Interrupt**. When working in the masked mode all internal breakpoints detected when MSR_{RI} =0 are lost and detected watchpoints are not counted by the debug counters. Detected watchpoints are always reported on the external pins, regardless of the value of the MSR_{RI} bit.

The core defaults to masked mode after reset. It is input in the nonmasked mode by setting the BRKNOMSK bit in the LCTRL2 register.The BRKNOMSK bit controls all internal breakpoints (I-breakpoints and L-breakpoints). See **Section 20.6.2.7 Load/Store Support AND-OR Control Register** for more information.

20.3.1.4 IGNORE FIRST MATCH OPTION. To facilitate the debugger utilities of "continue" and "go from x", the ignore first match option is supported for the instruction breakpoints. When an instruction breakpoint is first enabled, the first instruction will not cause an instruction breakpoint if the IFM bit in the instruction support control (ICTRL) register is set. This is used for "continue" utilities. When IFM is clear, every matched instruction can cause an instruction breakpoint. This is used for "go from x". The IFM bit is set by the software and cleared by the hardware after the first instruction breakpoint, the match is ignored. Load/store breakpoints and all counter-generated breakpoints (instruction and load/store) are unaffected by this mode.

20.3.1.5 GENERATING COMPARE TYPES. Using the four compare types—equal to, not equal to, greater than, and less than—it is possible to generate two additional compare types:

- Greater than or equal to
- Less than or equal to

Generating the greater than or equal to compare type can be accomplished by using the greater than compare type and programming the comparator to the value in question minus 1. Likewise, generating the less than or equal to compare type can be accomplished by using the less than compare type and programming the comparator to the value in question plus 1. Notice that this method does not work for the following boundary cases:

- Less than or equal to the largest unsigned number (1111...1)
- Greater than or equal to the smallest unsigned number (0000...0)
- Less than or equal to the maximum positive number when in signed mode (0111...1)
- Greater than or equal to the maximum negative number when in signed mode (1000...)

These boundary cases do not require special support because they are considered 'always true'. They can be programmed using the ignore option of the load/store watchpoint programming. See **Section 20.6.2.7 Load/Store Support AND-OR Control Register** for more information.

20.3.2 Basic Operation

20.3.2.1 INSTRUCTION SUPPORT. There are four instruction address comparators (A, B, C, and D). Each one is 30 bits long and generates two output events—equal to and less than. These signals generate one of four events—equal to, not equal to, greater than, or less than. The instruction watchpoints and breakpoint are generated using these events according to your programming. Using the OR option enables "out of range" detect.

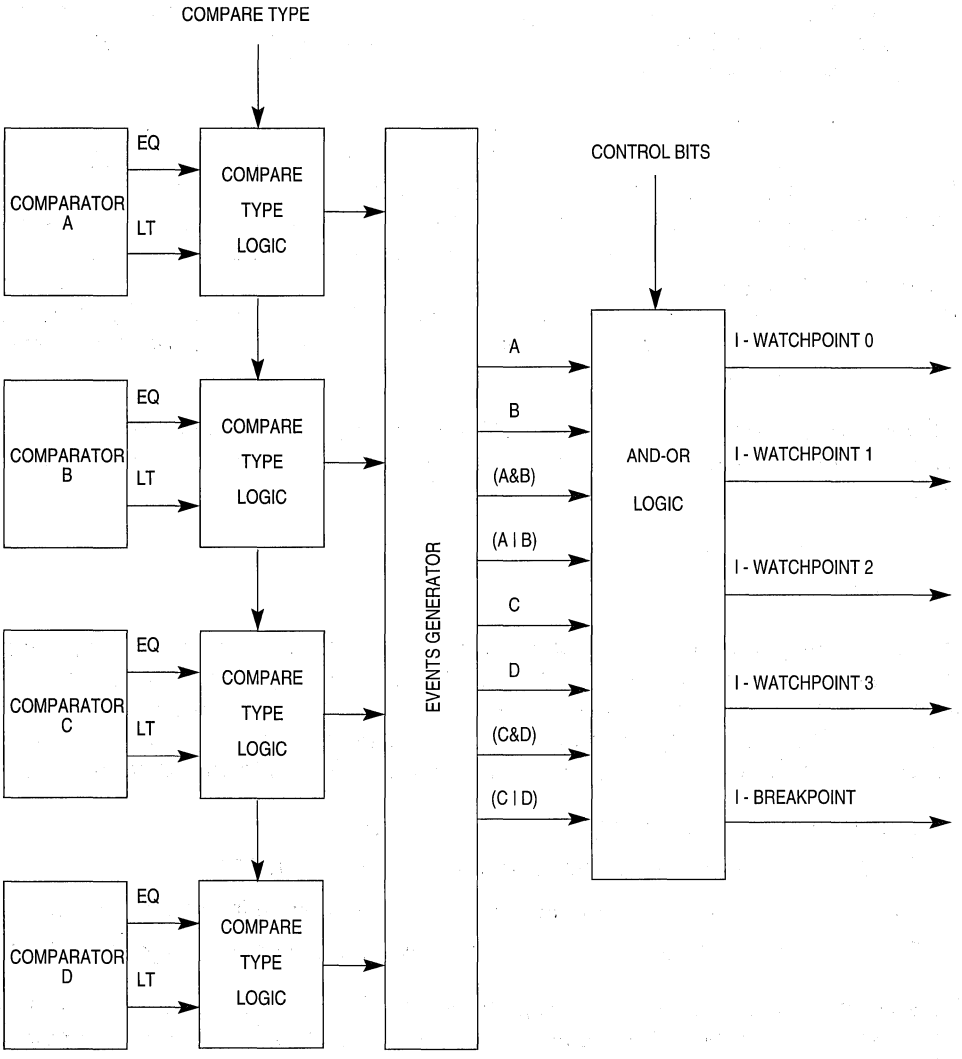


Figure 20-3. Instruction Support General Structure

Table 20-4. Instruction Watchpoints Programming Options

NAME	DESCRIPTION	PROGRAMMING OPTIONS
IW0	First instruction watchpoint	Comparator A Comparators (A & B)
IW1	Second instruction watchpoint	Comparator B Comparator (A | B)
IW2	Third instruction watchpoint	Comparator C Comparators (C & D)
IW3	Fourth instruction watchpoint	Comparator D Comparator (C | D)

20.3.2.2 LOAD/STORE SUPPORT. There are two load/store address comparators E and F that compare the 32 address bits and the cycle's attributes (read/write). The two least-significant bits are masked ignored whenever a word is accessed and the least-significant bit is masked whenever a half-word is accessed. Each comparator generates two output signals—equal to and less than. These signals generate one of four events from each comparator—equal to, not equal to, greater than, or less than. For more information, refer to **Section 20.3.1.2 Byte And Half-Word Working Modes**.

There are two load/store data comparators G and H that are 32 bits wide and can be programmed to treat numbers as signed or unsigned values. Each data comparator operates as four independent byte comparators that have a mask bit and generate two output signals—equal to and less than (if the mask bit is not set.) Therefore, each 32-bit comparator has eight output signals. These signals generate the "equal to and less than" signals according to the compare size that you program (byte, half-word, word). When operating in byte mode, all signals are significant. In half-word mode only four signals from each comparator are significant and in word mode only two signals are significant. In half-word mode only four signals from each comparator are significant, and in word mode only two signals are significant.

One of the following four match events are generated by the equal to and less than signals—equal to, not equal to, greater than, or less than—depending on the programmed compare type. Therefore, from the two 32-bit comparators, eight match indications are generated—Gmatch[0:3] and Hmatch[0:3]. According to the lower bits of the address and the size of the cycle, only match indications detected on bytes with valid information are validated. The rest are negated. If the executed cycle has a smaller size than the compare size (a byte access when the compare size is word or half-word), no match indication will be asserted. Using the match indication signals, four load/store data events are generated as shown in Table 20-5.

Table 20-5. Load/Store Data Events

EVENT NAME	EVENT FUNCTION
G	(Gmatch0 I Gmatch1 I Gmatch2 I Gmatch3)
H	(Hmatch0 I Hmatch1 I Hmatch2 I Hmatch3)
(G & H)	((Gmatch0 & Hmatch0) I (Gmatch1 & Hmatch1) I (Gmatch2 & Hmatch2) I (Gmatch3 & Hmatch3))
(G I H)	((Gmatch0 I Hmatch0) I (Gmatch1 I Hmatch1) I (Gmatch2 I Hmatch2) I (Gmatch3 I Hmatch3))

NOTE: & denotes a logical AND, but I denotes a logical OR.

The four load/store data events, combined with the match events of the load/store address comparators and the instruction watchpoints, are used to generate the load/store watchpoints and breakpoint according to your programming.

Table 20-6. Load/Store Watchpoints Programming Options

NAME	DESCRIPTION	I-ADDRESS EVENT PROGRAMMING OPTIONS	L-ADDRESS EVENT PROGRAMMING OPTIONS	L-DATA EVENT PROGRAMMING OPTIONS
LW0	First Load/Store Watchpoint	IW0, IW1, IW2, IW3, Ignore I-address events	Comparator E Comparator F Comparators (E & F) Comparators (E I F) Ignore L-address events	Comparator G Comparator H Comparators (G & H) Comparators (G I H) Ignore L-data Events
LW1	Second Load/ Store Watchpoint	IW0, IW1, IW2, IW3, Ignore I-address events	Comparator E Comparator F Comparators (E & F) Comparators (E I F) Ignore L-address events	Comparator G Comparator H Comparators (G & H) Comparators (G I H) Ignore L-data Events

When programming the load/store watchpoints to ignore L-address and L-data events, the instruction must be a load/store instruction to trigger the load/store watchpoint event.

20.3.2.3 COUNTER SUPPORT. There are two 16-bit down counters that count one of the instruction watchpoints or one of the load/store watchpoints. Both generate the corresponding breakpoint when they reach zero. When working in masked mode, the counters do not count detected watchpoints when MSR_{RI} =0. Counter values are not predictable if they are counting watchpoints programmed on the instructions that alter the counters. Readings from the active counters must be synchronized by inserting a **sync** instruction before a read is performed. For details, refer to **Section 20.3.1.2 Byte And Half-Word Working Modes**.

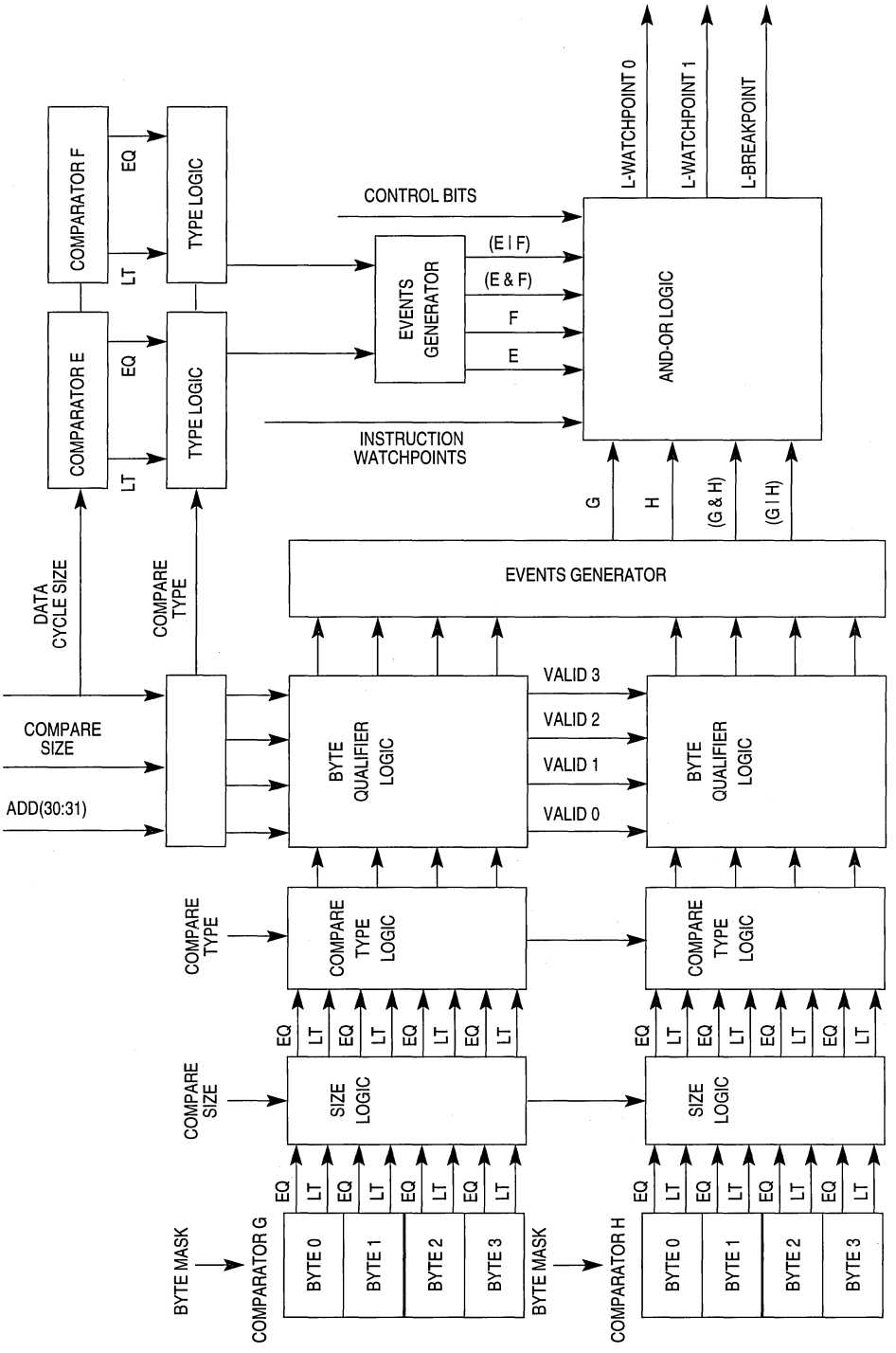


Figure 20-4. Load/Store Support General Structure

Note: When programmed to count instruction watchpoints, the last instruction that decrements the counter to zero is treated like any other instruction breakpoint in the sense that it is not executed before the machine branches to the breakpoint exception routine. As a side effect of this behavior, the value of the counter inside the breakpoint exception routine equals 1 and not zero. When programmed to count load/store watchpoints, the last instruction that decrements the counter to zero is treated like any other load/store breakpoint in the sense that it is executed before the machine branches to the breakpoint exception routine. Therefore, the value of the counter inside the breakpoint exception routine equals zero.

20.3.2.4 TRAP ENABLE PROGRAMMING. The trap enable bits can be programmed by regular software (only if $MSR_{PR} = 0$) using the **mtspr** instruction or on-the-fly using the special development port interface. For more information, refer to **Section 20.4.3.7 Trap Enable Mode**. The value used by the breakpoint generation logic is the bit-wise OR of the software trap enable bits written using the **mtspr** instruction and the development port trap enable bits that are serially shifted using the development port. The software trap enable bits and development port trap enable bits can be read from the ICTRL and LCTRL2 registers using the **mtspr** instruction. For exact bit placement, refer to **Section 20.6.2.7 Load/Store Support AND-OR Control Register** and **Section 20.6.2.5 Instruction Support Control Register**.

20.4 HARDWARE DEVELOPMENT SYSTEM INTERFACE

When debugging an existing system it is sometimes helpful to be able to do so without making any changes. Although, in some cases it is not helpful and may even make it impossible to add load to the lines connected to the existing system. The development system interface of the core supports this configuration.

The development system interface of the core uses the development port, which is a dedicated serial port that does not need any of the regular system interfaces. System activity can be controlled from the development port when the core is in debug mode. The development port is a relatively inexpensive interface that allows the development system to operate in a lower frequency than the core's frequency. It is also possible to debug the core using monitor debugger software. For more information, refer to **Section 20.5 Software Monitor Debugger**.

In debug mode, the core fetches all instructions from the development port. Data can be read from or written to the development port. This allows memory and registers to be read and modified by a development tool (emulator) connected to the development port. For protection purposes two possible working modes are defined—debug mode enable and debug mode disable. These working modes are only selected during reset. For details, refer to **Section 20.4.2.1 Debug Mode Enable vs. Debug Mode Disable**.

You can work in debug mode directly out of reset or the core can be programmed to enter into the debug mode as a result of a predefined sequence of events. These events can be any interrupt or exception in the core system (including the internal breakpoints) in addition to two levels of development port requests and one peripheral breakpoint request generated internally and externally. Each of these can be programmed as a regular interrupt that causes the machine to branch to its interrupt vector or as a special interrupt that causes debug mode entry. When in debug mode, the **rfi** instruction returns the machine to its regular work mode. The relationship between debug mode logic and the rest of the core is illustrated in the following figure.

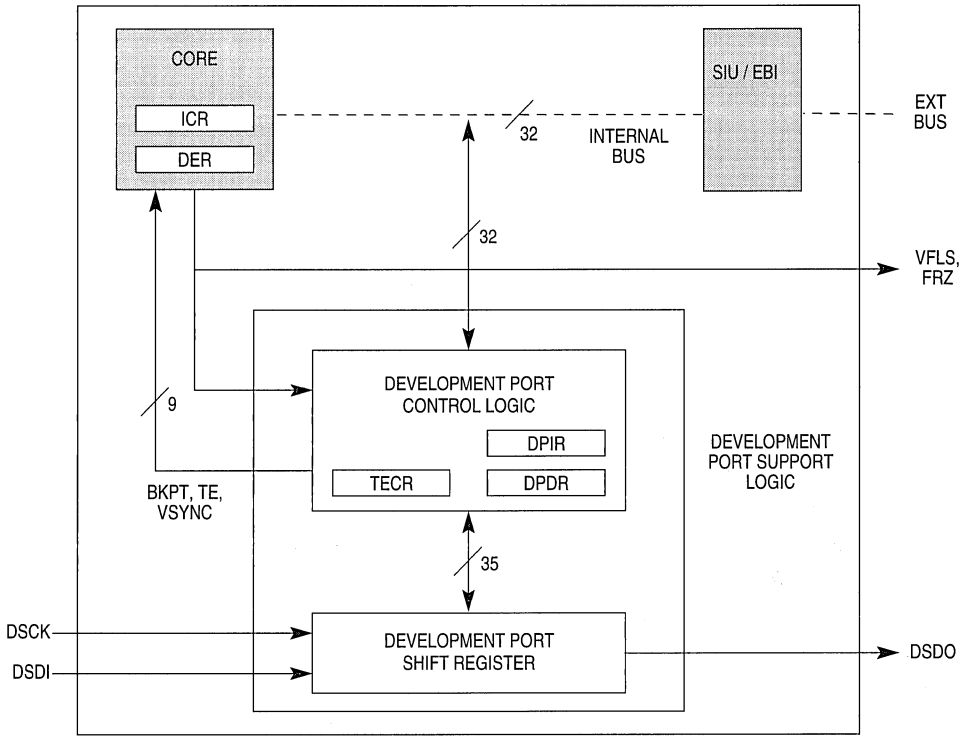


Figure 20-5. Relationship Between the CPU and Debug Mode

The development port provides a full-duplex serial interface for communications between the internal development support logic of the core and an external development tool. The development port can operate in two working modes—trap enable mode and debug mode.

20.4.1 Trap Enable Mode

The trap enable mode allows the following incidents to transfer control into the core internal development support logic.

- An instruction trap enable signal is used to program the instruction breakpoint on-the-fly.
- A load/store trap enable signal is used to program the load/store breakpoint on-the-fly.
- A nonmaskable breakpoint is used to assert the nonmaskable external breakpoint.
- A maskable breakpoint is used to assert the maskable external breakpoint.
- A VSYNC control code is used to assert and negate VSYNC operation.

In debug mode, the development port also controls the debug mode features of the core. For more details, refer to **Section 20.4.3 The Development Interface Port**.

20.4.2 Debug Mode

Debug mode provides the development system with the following functions:

- Controls and maintains all circumstances of processor execution.
- The development port can force the core to enter debug mode even when the external interrupts are disabled.
- Debug mode can be entered immediately out of reset, thus allowing you to debug a system without ROM.
- The events that cause the machine to enter into debug mode can be selectively defined through an enable register.
- Contains a cause register that indicates why debug mode is entered.
- After entering debug mode, program execution continues at the location where it first entered debug mode.
- All instructions are fetched from the development port, while load/store accesses are performed on the real system memory in debug mode.
- A simple method is provided for memory dump and load via the data register of the development port that is accessed with **mtspr** and **mfspr**.
- The processor enters the privileged state (MSR_{PR} =0) in debug mode, thus allowing execution of any instruction and access to any storage location.
- An OR signal of all interrupt cause register bits enables the development port to detect pending events while already in debug mode. For example, the development port can detect a debug mode access to nonexisting memory space.

Figure 20-6 illustrates the debug mode logic implemented in the core.

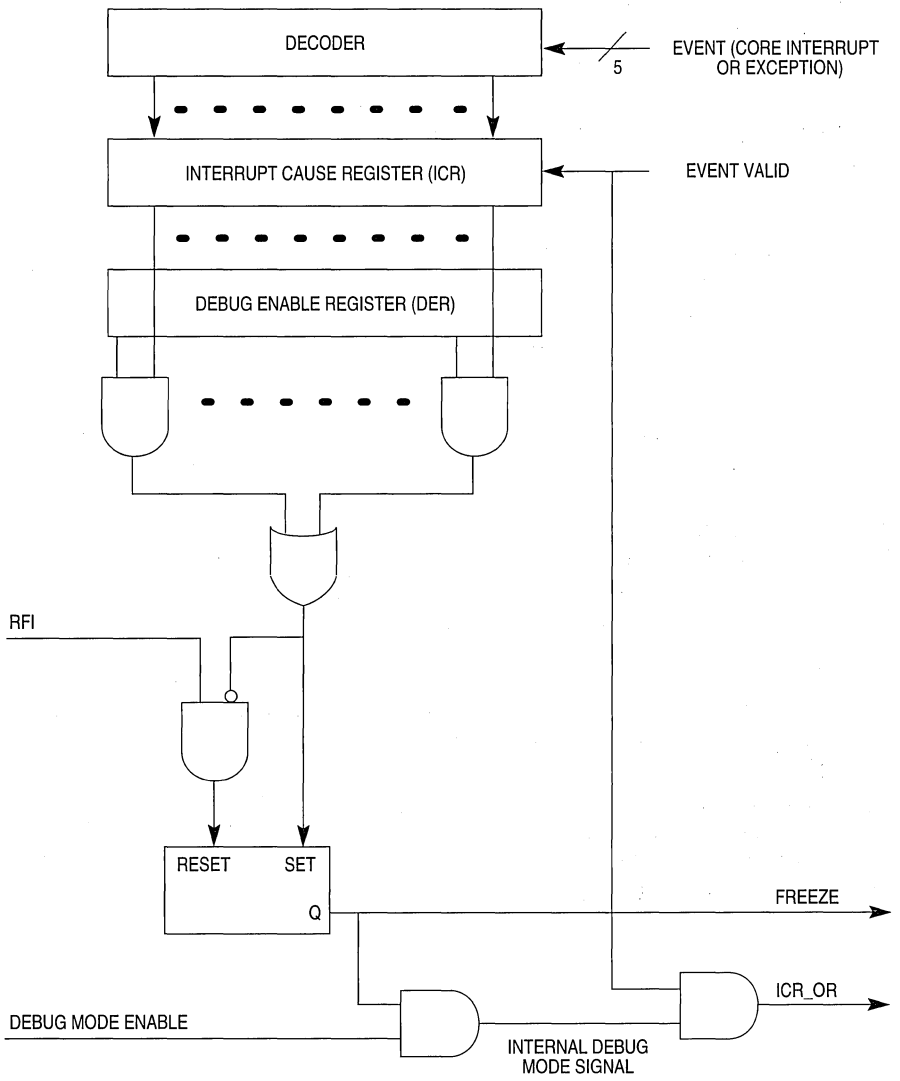


Figure 20-6. Debug Mode Logic Implementation

20.4.2.1 DEBUG MODE ENABLE VS. DEBUG MODE DISABLE. For protection purposes, there are two possible working modes—debug mode enable and debug mode disable. These modes are selected once at reset. Debug mode is enabled by asserting the DSCK pin during reset and the state of this pin is sampled three clocks before $\overline{\text{SRESET}}$ negation. If the DSCK pin is sampled negated, debug mode is disabled until a subsequent reset that occurs when the DSCK pin is asserted. When debug mode is disabled, the internal watchpoint/breakpoint hardware is still operational and can be used by a software monitor program for debugging purposes. A timing diagram for the enabling debug mode is illustrated in Figure 20-7

Note: $\overline{\text{SRESET}}$ negation time depends on an external pull-up resistor, so any reference to $\overline{\text{SRESET}}$ negation time refers to the time the MPC823 releases $\overline{\text{SRESET}}$. If the rise time of $\overline{\text{SRESET}}$ is long because of a large resistor, the setup time for the debug port signals should be adjusted accordingly.

When debug mode is disabled, all development support registers are accessible when MSR_{PR} =0 and can be used by the monitor debugger software. However, the processor never enters debug mode and the ICR and DER are only used to assert or negate the freeze signal. For more information on the software monitor debugger, refer to **Section 20.5 Software Monitor Debugger**. Only when the core is in debug mode, are all development support registers accessible. Therefore, the development system has full control of the core's development support features. For more information, see Table 20-12.

20.4.2.2 ENTERING DEBUG MODE. Debug mode entry can be the result of a number of events. All events have a programmable enable bit so you can selectively decide that the cause of debug mode entry as well as the events that require regular interrupt handling. Entering debug mode is possible immediately out of reset, thus allowing a system to be debugged without ROM. Specially programming the development port during reset makes this possible. If the DSCK pin is asserted during $\overline{\text{SRESET}}$ assertion and after $\overline{\text{SRESET}}$ negation, the processor will take a breakpoint exception and go directly to debug mode instead of fetching the reset vector.

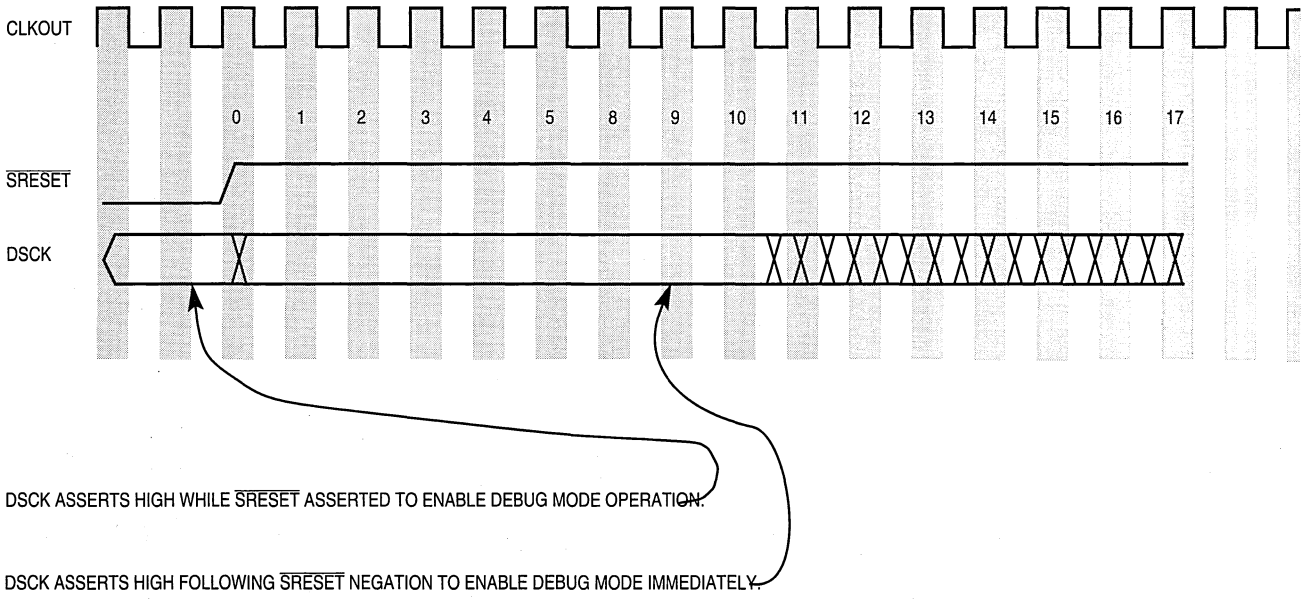


Figure 20-7. Debug Mode Reset Configuration Timing Diagram

To avoid entering debug mode after reset, the DSCK pin must be negated no later than seven clock cycles after $\overline{\text{SRESET}}$ negates to allow the processor to jump to the reset vector and begin normal execution. Entering debug mode immediately after reset, Bit 31 (development port interrupt bit) of ICR is set. For details, refer to the timing diagram illustrated in Figure 20-7.

When debug mode is disabled, all events result in regular interrupt handling. The internal freeze signal is asserted whenever an enabled event occurs, regardless of whether or not debug mode is enabled or disabled. The internal freeze signal is connected to all relevant internal modules. These modules can be programmed to stop all operations in response to the assertion of the freeze signal. For more information, refer to **Section 20.5.1 Freeze Indication (FRZ)**. Furthermore, the freeze indication is negated when exiting the debug mode and **Section 20.4.2.6 Exiting Debug Mode** has more information on the issue.

The following list of events can cause the core to enter debug mode. Each event results in debug mode entry if debug mode is enabled and the corresponding enable bit is set. The reset values of the enable bits allow debug mode features to be used even when debug enable mode (DER) is not programmed. For more information, see **Section 20.6.3.2 Debug Enable Register**.

- System reset as a result of $\overline{\text{SRESET}}$ assertion
- Checkstop
- Machine check interrupt
- Implementation specific instruction TLB miss
- Implementation specific instruction TLB error
- Implementation specific data TLB miss
- Implementation specific data TLB error
- External interrupt, recognized when MSR_{EE} =1
- Alignment interrupt
- Program interrupt
- Floating-point unavailable interrupt
- Decrementer interrupt, recognized when MSR_{EE} =1
- System call interrupt
- Trace asserted when in single or branch trace mode
- Implementation dependent software emulation interrupt
- Instruction breakpoint is recognized only when MSR_{RI} =1 and when breakpoints are masked. When breakpoints are not masked, they are always recognized.
- Load/store breakpoint is recognized only when MSR_{RI} = 1 and when breakpoints are masked. When breakpoints are not masked, they are always recognized.

- Peripheral breakpoint from the development port generated by external modules are recognized only when MSR_{RI} =1.
- Development port nonmaskable interrupt occurs as a result of a debug station request. Useful in some catastrophic events like an endless loop when MSR_{RI} =0. As a result of this event, the machine can enter a nonrestartable state.

The processor enters into the debug mode state when at least one of the bits in the ICR is set, the corresponding bit in the DER is enabled, and debug mode is enabled. When debug mode is enabled and an enabled event occurs, the processor waits until its pipeline is empty and then starts fetching the next instructions from the development port. For information on the exact value of the SRR0 and SRR1 registers, refer to **Section 7.3.7.3 Definitions**. When the processor is in debug mode, the freeze indication is asserted, thus allowing any properly programmed peripheral to stop. The fact that the core is in debug mode is also broadcasted to the external world using the value b'11' on the VFLS pins. The freeze signal can be asserted by the software when debug mode is disabled. The development port should read the value of the ICR to find out what causes debug mode entry. Reading the ICR clears all of its bits.

20.4.2.3 CHECKSTOP STATE AND DEBUG MODE. The core enters checkstop state if the machine check interrupt is disabled (MSR_{ME} =0) and a machine check interrupt is detected. However, if a machine check interrupt is detected when MSR_{ME} =0, debug mode is enabled, the checkstop enable bit in the DER is set, and the core enters debug mode rather then the checkstop state. The various actions taken by the core when a machine check interrupt is detected are provided in the following table.

Table 20-7. Checkstop State and Debug Mode

DEBUG MODE ENABLE	MSR_{ME}	CHSTPE[1]	MCIE[2]	ACTION PERFORMED BY THE CORE WHEN A MACHINE CHECK INTERRUPT IS DETECTED	ICR VALUE
0	0	X	X	Enter Checkstop State	0x20000000
0	1	X	X	Branch to the Machine Check Interrupt	0x10000000
1	0	0	X	Enter Checkstop State	0x20000000
1	0	1	X	Enter Debug Mode	0x20000000
1	1	X	0	Branch to the Machine Check Interrupt	0x10000000
1	1	X	1	Enter Debug Mode	0x10000000

NOTES:

1. The checkstop enable bit of the DER register.
2. The machine check interrupt enable bit of the DER register.

20.4.2.4 SAVING THE MACHINE STATE IN DEBUG MODE. If entering debug mode is the result of a load/store-type exception, the DAR and DSISR registers contain critical information. These two registers must be saved before any other operation is performed. Failing to save these registers can result in information loss if another load/store-type exception occurs inside the development software. Since exceptions are treated differently in debug mode, there is no need to save the SRR0 and SRR1 registers.

20.4.2.5 RUNNING IN DEBUG MODE. When running in debug mode, all fetch cycles access the development port, regardless of the cycle's actual address. All load/store cycles access the real memory system according to the cycle's address. The data register of the development port is mapped as a special control register and is accessed using the **mtspr** and **mfspr** instructions, via special load/store cycles.

Exceptions are treated differently in debug mode. When in debug mode, the ICR is updated when an exception is recognized by the event that caused the exception. A special error indication (ICR_OR) is asserted for one clock cycle to notify the development port that an exception has occurred. Execution then continues in debug mode without any change in the SRR0 and SRR1 registers. ICR_OR is asserted before the next fetch occurs so the development system can detect the excepting instruction. However, not all exceptions are recognizable in debug mode. Breakpoints and watchpoints are not generated by the hardware when in debug mode, regardless of the MSR_{RI} bit's value. When entering debug mode, the MSR_{EE} bit is cleared by the hardware, thus forcing the hardware to ignore external and decrementer interrupts.

Caution: Setting the MSR_{EE} bit with the debug software in debug mode is strictly forbidden.

This restriction is relevant because the external interrupt event is a level signal. Because the core only reports exceptions in debug mode and does not perform exception processing, the core hardware does not clear the MSR_{EE} bit. This event, if enabled, is then recognized on every clock. When the ICR_OR signal is asserted, the development station must search the ICR to find the event that caused the exception. Since the values in the SRR0 and SRR1 registers do not change if an exception is recognized in debug mode, they only change once when entering debug mode. However, it is not necessary to save the SRR0 and SRR1 registers when entering debug mode.

20.4.2.6 EXITING DEBUG MODE. The **rfi** instruction is used to exit from debug mode and return to normal processor operation and negate the FRZ signal. The development system may monitor the FRZ signal or status to make sure the MPC823 is out of debug mode. It is the responsibility of the software to read the ICR before performing the rfi instruction. Failure to do so forces the core to immediately reenter debug mode and reassert the freeze signal if an asserted bit in the ICR register has a corresponding enable bit set in the DER register.

20.4.3 The Development Interface Port

The development port provides a full-duplex serial interface for communications between the internal development support logic and an external development tool. The relationship of the development support logic to the rest of the core is illustrated in Figure 20-5. Notice that the development port support logic is shown as a separate block for clarity. It will be implemented as part of the system interface unit module. The development interface port contains the four pins:

- Development serial clock
- Development serial data in
- Development serial data out
- Freeze

20.4.3.1 DEVELOPMENT SERIAL CLOCK. The development serial clock (DSCK) pin is used to shift data into and out of the development interface port shift register. At the same time, the new most-significant bit of the shift register is presented to the development serial data out (DSDO) pin. Future references to the DSCK signal imply the internally synchronized value of the clock. The DSCK input must be driven either high or low at all times and it not allowed to float. With a resistor, a typical target environment would pull this input low.

The clock can be implemented as a free-running or gated clock. The shifting of data is controlled by the ready and start signals, so the clock does not need to be gated with the serial transmissions. The DSCK pin is used at reset to enable debug mode either immediately following reset or to enter debug mode driving an event.

20.4.3.2 DEVELOPMENT SERIAL DATA IN. Data to be transferred into the development interface port shift register is presented to the development serial data in (DSDI) pin by external logic. When driven asynchronous with the system clock, the data presented to the DSDI pin must be stable at setup time before the rising edge of DSCK and at hold time after the rising edge of DSCK. When synchronously driven to the system clock, the data must be stable on DSDI or a setup time before system clock output (CLKOUT) rising edge and a hold time after the rising edge of CLKOUT. The DSDI pin is also used at reset to control the overall chip configuration mode and determine the development port clock mode. Refer to **Section 20.4.3.6 Development Port Serial Communication** for more information.

20.4.3.3 DEVELOPMENT SERIAL DATA OUT. The debug mode logic shifts data out of the development interface port shift register using the development serial data out (DSDO) pin. All transitions on DSDO are synchronous with DSCK or CLKOUT, depending on the clock mode. Data will be valid at setup time before the rising edge of the clock and remains valid at hold time after the rising edge of the clock. See Table 20-10 for details about DSDO data.

20.4.3.4 FREEZE. A freeze indication means that the processor is in debug mode and that normal processor execution of user code is frozen. The freeze state is indicated on the FRZ pin and is generated synchronous to the system clock. This indication can be used to halt any off-chip device while in debug mode and is a handshake between the debug tool and port. In addition to the FRZ pin, the freeze state is indicated by the value b11 on the VFLS pins. The internal freeze status can also be monitored through status in the data shifted out of the debug port.

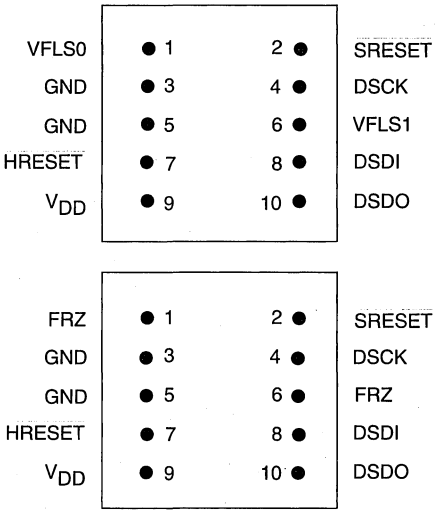


Figure 20-8. Development Port/Background Development Mode Connector Pinout Options

20.4.3.5 DEVELOPMENT INTERFACE PORT REGISTERS. The development interface port consists logically of three registers—development interface port instruction register, development interface port data register, and trap enable control register. However, these registers are physically implemented as two registers—the development interface port shift register and the trap enable control register. The development interface port shift register acts as both the DPIR and DPDR, depending on the operation being performed. It is also used as a temporary holding register for data to be stored in the TECR. See Table 6-9 for more information.

20.4.3.5.1 Development Interface Port Shift Register. The development interface port shift register (DPIR/DPDR) is a 35-bit shift register. Instructions and data are serially shifted into it from the DSDI using DSCK or CLKOUT as the shift clock, which depends on the debug port clock mode. For more information refer to **Section 20.4.3.6 Development Port Serial Communication**.

The instructions or data are then transferred in parallel to the core and TECR. When the processor enters debug mode it fetches instructions from the DPIR, which causes an access to the development interface port shift register. These instructions are serially loaded into the shift register from the DSDI using DSCK or CLKOUT as the shift clock. Similarly, data is transferred to the core. Data is shifted into the shift register and read by the processor when a "move from special-purpose register DPDR" instruction is executed. Data is also parallel loaded into the development interface port shift register from the core by executing a "move to special-purpose register DPDR" instruction. It is then serially shifted out to the DSDO pin using DSCK or CLKOUT as the shift clock.

20.4.3.5.2 Trap Enable Control Register. The 9-bit trap enable control register (TECR) is loaded from the development interface port shift register. The contents of the control register drive the six trap enable signals, two breakpoint signals, and the VSYNC signal to the core. The transfer data to trap enable control register commands are used to force the appropriate bits to be transferred to this register. The trap enable control register is not accessed by the core, but supplies signals to the core. The trap enable bits, VSYNC bit, and the breakpoint bits of this register are loaded from the development interface port shift register as a result of trap enable mode transmissions. The trap enable bits are reflected in the ICTRL and LCTRL2 special registers. Refer to **Section 20.6.2 Development Port Registers** for more information on the support registers.

20.4.3.5.3 Decoding the Development Interface Port Registers. The development interface port shift register is selected when the core accesses the DPIR or DPDR registers. Accesses to these two special-purpose registers occur in debug mode and appear on the internal bus as an address and the assertion of an address attribute signal indicating that a special-purpose register is being accessed. The DPIR is read by the core to fetch all instructions when in debug mode. The DPDR is read and written to transfer data between the core and external development tools. The DPIR and DPDR are pseudo-registers, so decoding either of these registers causes the development interface port shift register to be accessed. The debug mode logic knows whether the core is fetching instructions or reading or writing data. A sequence error is signaled to the external development tool when the core expected result and the GPR result do not match. An example of this would be when an instruction is received instead of the expected data.

20.4.3.6 DEVELOPMENT PORT SERIAL COMMUNICATION. All serial transmissions are synchronous, with respect to the transmission clock.

20.4.3.6.1 Clock Mode Selection. With respect to CLKOUT, the transmission clock can either be synchronous or asynchronous. The development port has two methods that can be used to clock serial transmissions. The first method allows the transmission to occur without being externally synchronized to CLKOUT. In this mode, a serial clock DSCK must be supplied to the MPC823. The other communication method requires data to be externally synchronized with respect to CLKOUT.

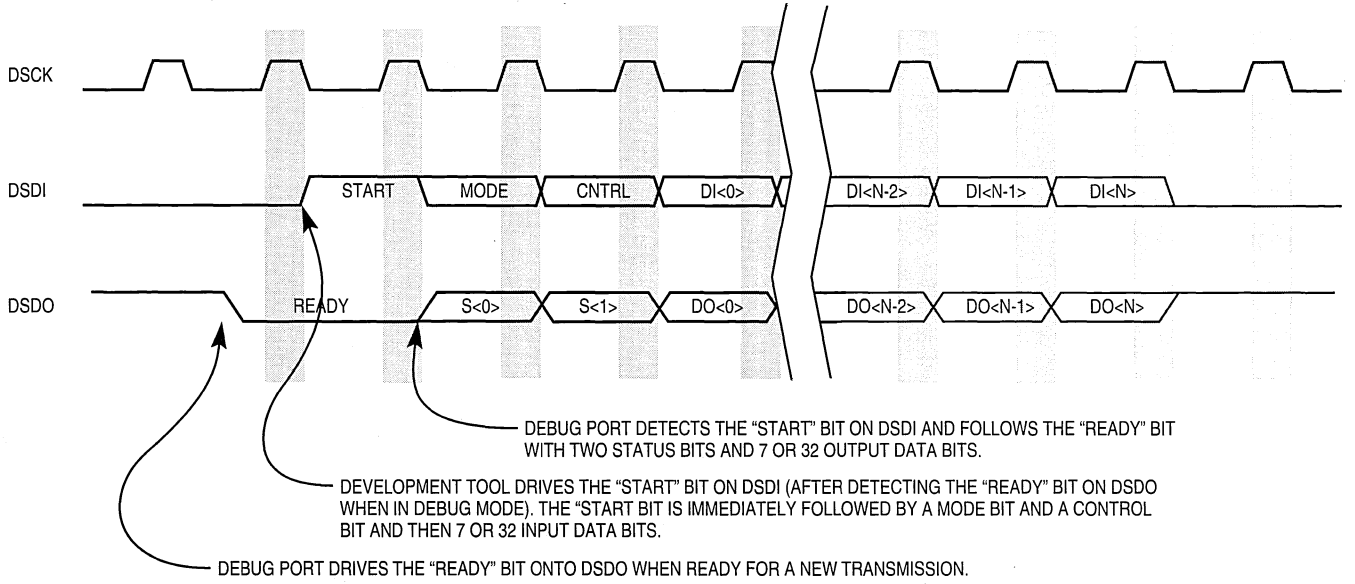
The first clock mode is called asynchronous clocked since the input clock DSCK is asynchronous with respect to CLKOUT. To be sure that data on DSDI is sampled correctly, transitions on DSDI must meet all setup and hold times in respect to the rising edge of DSCK. This clock mode allows communications with the port from a development tool that does not have access to the CLKOUT signal or has either a delayed or skewed CLKOUT signal. The timing diagram in Figure 20-9 illustrates serial communication asynchronous clocked timing.

The second clock mode is called synchronous self-clocked and does not require an input clock. Instead, the port is clocked by the system clock. The DSDI input is required to meet setup and hold time requirements, with respect to the rising edge of CLKOUT. The data rate for this mode is always the same as the system clock. The timing diagram in Figure 20-10 illustrates serial communication synchronous self-clocked timing. The selection of clocked or self-clocked mode is made at reset. The state of the DSDI input is latched eight clocks after $\overline{\text{SRESET}}$ is negated. If it is latched low, asynchronous clocked mode is enabled. If it is latched high, then synchronous self-clocked mode is enabled. The timing diagram in Figure 20-11 illustrates the clock mode selection following reset.

Since DSDI is used to select the development port clock scheme, any transitions on DSDI during clock mode select must be prevented from being recognized as the start of a serial transmission. The port will not begin scanning for the START bit of a serial transmission until 16 clocks after $\overline{\text{SRESET}}$ is negated. If DSDI is asserted 16 clocks after $\overline{\text{SRESET}}$ negates, the port waits until DSDI is negated before it starts scanning for the START bit.

20.4.3.7 TRAP ENABLE MODE. When not in debug mode, the development port begins communicating by setting DSDO (the MSB of the 35-bit development interface port shift register) low to show that all activity related to the previous transmission is complete and that a new transmission can begin. The start of a serial transmission from an external development tool to the development port is signaled by a START bit. A MODE bit in the transmission defines it as either a trap enable mode or debug mode transmission. If the MODE bit is set, the transmission will be 10 bits long and only seven data bits will be shifted into the shift register. These seven bits will be latched into the TECR. A control bit determines whether the data is latched into the TECR's trap enable, VSYNC, or breakpoints bits.

The development interface port shift register is 35 bits wide, but trap enable mode transmissions only use 10 of the 35 bits—start/ready, mode/status, control/status, and seven least-significant data bits. The encoding of data shifted into the development interface port shift register is shown in Table 20-8 and Table 20-9.



NOTE: DSCK AND DSDI TRANSITIONS ARE NOT REQUIRED TO BE SYNCHRONOUS WITH CLKOUT.

Figure 20-9. Asynchronous Clocked Serial Communications Timing Diagram

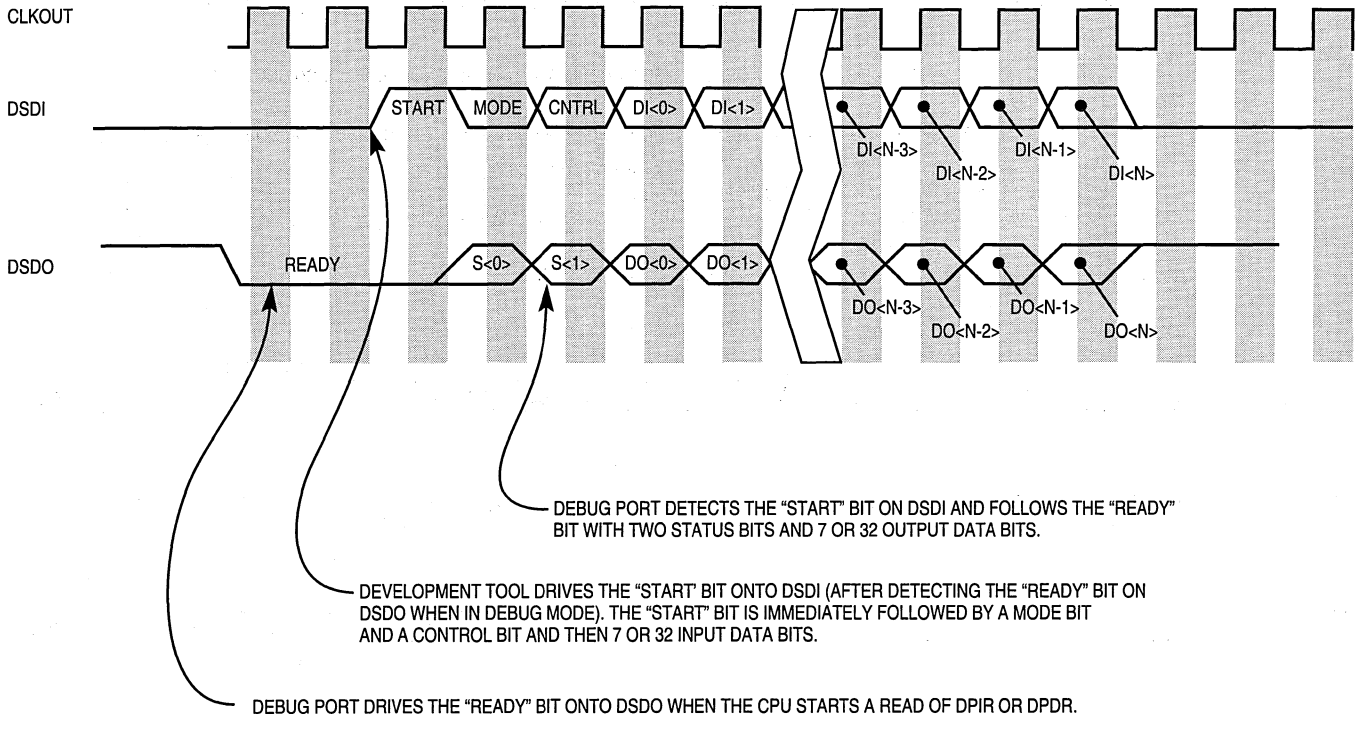


Figure 20-10. Synchronous Self-Clocked Serial Communications Timing Diagram

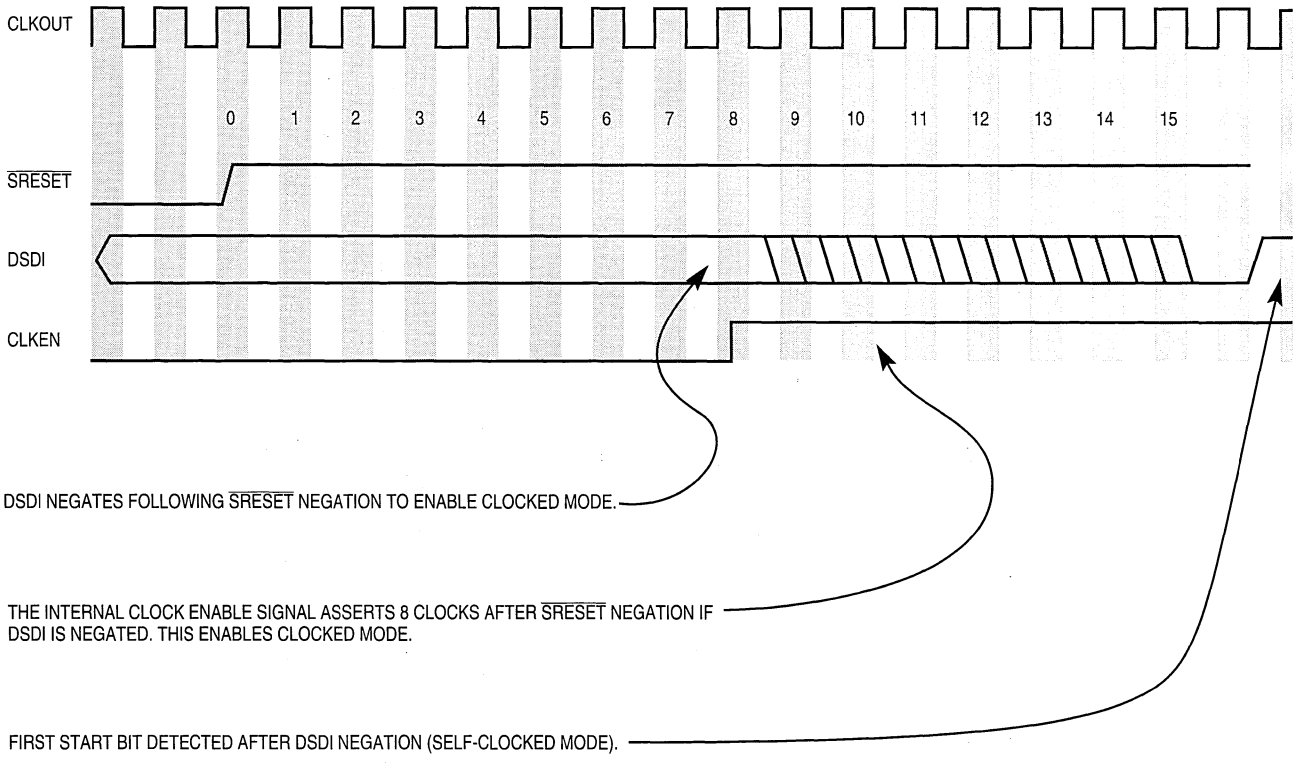


Figure 20-11. Enabling Clock Mode Following Reset Timing Diagram

Table 20-8. Trap Enable Data Shifted Into DPS Register

START	MODE	CONTROL	FIRST	SECOND	THIRD	FOURTH	FIRST	SECOND	VSYNC	FUNCTION
			INSTRUCTION WATCHPOINT TRAP ENABLES			DATA WATCHPOINT TRAP ENABLES				
1	1	0	0 = Disabled 1 = Enabled							Transfer Data to Trap Enable Control Register

Table 20-9. DEBUG PORT Command Shifted Into the DPS Register

START	MODE	CONTROL	EXTENDED OPCODE		MAJOR OPCODE	FUNCTION
1	1	1	x	x	00000	NOP
					00001	Hard Reset Request
					00010	Soft Reset Request
			0	x	00011	Reserved
			1	0	00011	End Download Procedure
			1	1	00011	Start Download Procedure
			x	x	00100 — 11110	Reserved
			x	0	11111	Negate Maskable Breakpoint
			x	1	11111	Assert Maskable Breakpoint
			0	x	11111	Negate Nonmaskable Breakpoint
			1	x	11111	Assert Nonmaskable Breakpoint

The watchpoint trap enable and VSYNC functions are described in **Section 20.3 Generating Watchpoints And Breakpoints** and **Section 20.2 Program Flow Tracking**. The **DEBUG PORT** command allows the development tool to either assert or negate breakpoint requests, reset the processor, or activate or deactivate the fast download procedure. Status out of the development interface port in the trap enable mode is shown in Table 20-10.

Table 20-10. Status/Data Shifted Out of DPS Register

READY	STATUS [0:1]		DATA			FUNCTION
			BIT 0	BIT 1	BITS 2-31 OR 2-6, DEPENDING ON THE INPUT MODE	
(0)	0	0	DATA			Valid Data From Core
(0)	0	1	Freeze Status	Download Procedure In Progress	1 sec	Sequencing Error
(0)	1	0			1 sec	Core Interrupt
(0)	1	1			1 sec	Null

NOTE: For freeze status, 0 means the core is in normal mode and 1 means the core is in debug mode. For download status, 0 means the download is in progress and 1 means the core is in normal mode.

In trap enable mode the "Valid Data from CPU" and "CPU Interrupt" status cannot occur. Out of debug mode, the sequencing error encoding indicates that the transmission from the external development tool was a debug mode transmission. When a sequencing error occurs, the development interface port ignores the data shifted in while the sequencing error is shifting out and being treated as a no operation (NOP) function. The null output encoding indicates that the previous transmission did not have any associated errors. Out of debug mode, ready will be asserted at the end of each transmission. If debug mode is not enabled and transmission errors are guaranteed not to occur, the status output is not needed.

20.4.3.8 DEBUG MODE. In debug mode the development interface port starts communicating by setting DSDO low to show that the core is trying to read an instruction from the DPIR or data from the DPDR. When the core writes data to the port to be shifted out, the READY bit is not set. The port waits for the core to read the next instruction before asserting ready. This allows duplex operation of the serial port while allowing the port to control all transmissions from the external development tool. After detecting this ready status, the external development tool begins the transmitting the development interface port with a START bit (logic high) on the DSDI pin.

In debug mode the 35 bits of the development interface port shift register are interpreted as a START/READY bit, a MODE/STATUS bit, a CONTROL/STATUS bit, and 32 bits of data. All instructions and data for the core are transmitted with the mode bit cleared, thus indicating a 32-bit DATA field. The encoding of data shifted into the development interface port shift register through the DSDI pin is shown in Table 20-11.

Table 20-11. Debug Instructions/Data Shifted Into the DPS Register

START	MODE	CONTROL	INSTRUCTION / DATA (32 BITS)		FUNCTION
			BITS 0:6	BITS 7:31	
1	0	0	CPU Instruction		Transfer Instruction to Core
1	0	1	CPU Data		Transfer Data to Core
1	1	0	Trap Enable Bits	Not Exist	Transfer Data to Trap Enable Control Register
1	1	1	0011111	Not Exist	Negate Breakpoint Requests to Core
1	1	1	0	Not Exist	NOP

NOTE: See Table 20-8 for details on trap enable bits and Table 20-9 for details on debug port commands.

All transmissions from the debug port on DSDO begin with a zero or ready bit. This indicates that the core is trying to read an instruction or data from the port. The external development tool waits until it sees DSDO go low before it starts sending the next transmission. The control bit differentiates between instructions and data that allow the development interface port to detect an instruction that was entered when the core was expecting data and vice versa. If this occurs, a sequence error indication is shifted out in the next serial transmission. The trap enable function allows the development interface port to transfer data to the trap enable control register. The **DEBUG PORT** command allows the development tool to either negate breakpoint requests, reset the processor, or activate or deactivate the fast download procedure. The NOP function provides a null operation to use when there is data or a response to be shifted out of the data register. The next appropriate instruction or command will be determined by the value of the response or data shifted out.

The encoding of data shifted out of the development interface port shift register in debug mode is the same as for trap enable mode, as shown in Table 20-10. The valid data encoding is used when data has been transferred from the core to the development interface port shift register. This results when an instruction to move the contents of a general-purpose register to the DPDR occurs. The valid data encoding has the highest priority of all status outputs and will be reported even if an interrupt occurs at the same time. Since it is not possible for a sequencing error to occur that has valid data, there is no priority conflict with the sequencing error status. Also, any interrupt that is recognized at the same time that there is valid data, is not related to the execution of an instruction. Therefore, a valid data status will be output and the interrupt status will be saved for the next transmission.

The sequencing error encoding indicates that the external development tool inputs are not what the development interface port and/or the core was expecting. There are two possible causes of this error:

- The processor was trying to read instructions and data was shifted into the development interface port.
- The processor was trying to read data and an instruction was shifted into the development interface port.

Nonetheless, the port terminates the read cycle with a bus error. In turn, this bus error causes the core to signal that an interrupt exception has occurred. Since a status of sequencing error is of higher priority than an exception, the port reports the sequencing error first and the core interrupt on the next transmission. The development interface port ignores the command, instruction, or data shifted in while the sequencing error or core interrupt is shifted out. The next transmission, after all error status is reported to the port, should be either a new instruction, trap enable, or command.

The interrupt encoding that has occurred indicates that the core encountered an interrupt while executing the previous instruction in debug mode. Interrupts can occur as the result of instruction execution (such as unimplemented opcode or arithmetic error), because of a memory access fault, or from an unmasked external interrupt. When an interrupt occurs, the development interface port ignores the command, instruction, or data shifted in while the interrupt encoding was shifting out. The next transmission to the port should be a new instruction, trap enable, or **DEBUG PORT** command. Finally, the null encoding indicates that no data has been transferred from the core to the development interface port shift register.

The fast download procedure is used to download a block of data from the debug tool into the system memory. This procedure can be accomplished by repeating the following sequence of transactions from the development tool to the debug port for the number of data words to be downloaded.

```
INIT:Save RX, RY
RY <- Memory Block address- 4

...

repeat:mfspr RX, DPDR
DATA word to be moved to memory
stwu RX, 0x4(RY)
until here

...

Restore RX,RY
```

Figure 20-12. Download Procedure Code Example

For large blocks of data, this sequence can take a significant amount of time to complete. Using the fast download procedure of the debug port will reduce the time by eliminating the need to transfer the instructions in the loop to the debug port. The only transactions needed are those used to transfer the data to be placed in the system memory. Figure 20-13 and Figure 20-14 illustrate the benefit of using the fast download procedure.

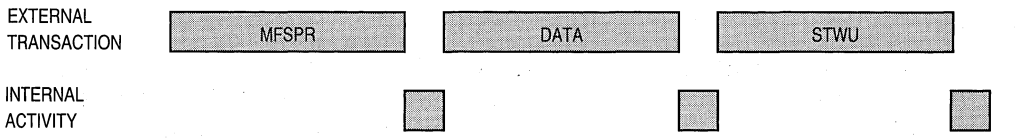


Figure 20-13. Slow Download Procedure Loop

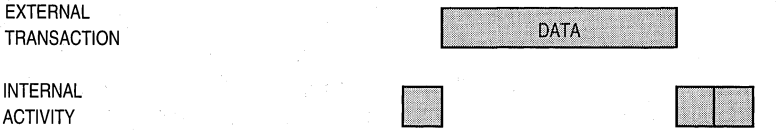


Figure 20-14. Fast Download Procedure Loop

The sequence of the instructions used in the fast download procedure is illustrated in Figure 20-12, with RX = r31 and RY = r30. This sequence is repeated infinitely until the **END DOWNLOAD PROCEDURE** command is issued to the debug port. The internal general-purpose register 31 is used for temporary storage of the data value. Before beginning the fast download procedure by issuing the **START DOWNLOAD PROCEDURE** command, the value (of the first memory block address -4) must be written into the general-purpose register 30. To end the download procedure, an **END DOWNLOAD PROCEDURE** command should be issued to the debug port and the development tool will send an additional data transaction. This data word will not be placed into the system memory, but it is needed to stop the procedure.

20.5 SOFTWARE MONITOR DEBUGGER

When in debug mode disable, a software monitor debugger can use all of the development support features defined in the core. When debug mode is disabled, all events result in regular interrupt handling in which the processor resumes execution in the corresponding interrupt handler. The ICR and DER only influence the assertion and negation of the freeze signal.

20.5.1 Freeze Indication (FRZ)

The internal freeze signal is connected to all relevant internal modules that can be programmed to stop all operations in response to freeze signal assertion. To enable a software monitor debugger to signal that the debug software is now executed, the internal freeze signal can be asserted or negated when debug mode is disabled.

Assertion of the freeze signal is broadcasted to the external world over FRZ. Asserting and negating the freeze signal when in disabled debug mode is controlled by the ICR and DER, as illustrated in Figure 20-6.

To assert the FRZ signal, the software must be programmed to use the relevant bits in the DER. To negate the FRZ signal the software must read the ICR to clear it and perform an **rfi** instruction. If the ICR is not cleared before the **rfi** instruction is performed, the FRZ signal is not negated. Therefore, nested exception tracing can be supported by the software monitor debugger without affecting the value of the FRZ signal. Only before the last **rfi** instruction does the software need to clear the ICR. This process enables the software to accurately control FRZ assertion or negation.

20.6 PROGRAMMING THE DEVELOPMENT PORT REGISTERS

Normally, the development port registers reside in the control register space and can be accessed using the **mtspr** and **mfspr** instructions. They also reside in the memory map I/o to be accessed by the **ld** and **st** instructions. The addresses of these registers are in Table 6-9.

20.6.1 Protecting the Development Port Registers

The development support registers are protected as shown in the following table. Take note of the ICR and DPDR registers' special behavior.

Table 20-12. Development Support Register Protection

OPERATION	MSR_{PR}	DEBUG MODE ENABLE	IN DEBUG MODE	RESULT
Read Register	0	0	X	A read is performed and when reading ICR, it is also cleared.
	0	1	0	A read is performed and when reading ICR, it is not cleared.
	0	1	1	A read is performed and when reading ICR, it is also cleared.
	1	X	X	A read is not performed, a program interrupt is generated, and when reading ICR, it is not cleared.
Write Register	0	0	X	A write is performed, a write to ICR is ignored, and a write to DPDR is ignored.
	0	1	0	A write is ignored.
	0	1	1	A write is performed and a write to ICR is ignored.
	1	X	X	A write is not performed, but a program interrupt is generated.

20.6.2 Development Port Registers

20.6.2.1 COMPARATOR A–D VALUE REGISTERS. The comparator A-D value (CMPA-D) registers contain the address to be compared to the instruction address for generating an instruction watchpoint. These registers have an undefined reset value.

CMPA-D

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	ICMPV															
RESET	X															
R/W	R/W															
SPR	144, 145, 146, 147															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	ICMPV														RESERVED	
RESET	X														—	
R/W	R/W														R/W	
SPR	144, 145, 146, 147															

NOTE: X = "Don't Care".

ICMPV—Instruction Comparison Value
This field represents the address bits to be compared to the instruction address.

Bits 30–31—Reserved
These bits are reserved and should be set to 0.

20.6.2.2 COMPARATOR E–F VALUE REGISTERS. The comparator E-F value (CMPE-F) registers contain the address to be compared to the load/store address for generating a load/store watchpoint. These registers have an undefined reset value.

CMPE-F

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	LCMPV															
RESET	X															
R/W	R/W															
SPR	152, 153															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	LCMPV															
RESET	X															
R/W	R/W															
SPR	152, 153															

NOTE: X = "Don't Care".

LCMPV—Load/Store Comparison Value

This field is the address bits to be compared to the load/store address.

20.6.2.3 COMPARATOR G–H VALUE REGISTERS. The comparator G-H value (CMPG-H) registers contain the data to be compared to the translation data. These registers have an undefined reset value.

CMPG-H

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	DCMPV															
RESET	X															
R/W	R/W															
SPR	154, 155															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	DCMPV															
RESET	X															
R/W	R/W															
SPR	154, 155															

NOTE: X = "Don't Care".

DCMPV—Data Comparison Value

This field represents data to be compared to the translation data.

20.6.2.4 BREAKPOINT ADDRESS REGISTER. The breakpoint address (BAR) register contains the address of the transaction or the instruction that causes the breakpoint. This register has an undefined reset value.

BAR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	BARV															
RESET	X															
R/W	R/W															
SPR	159															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	BARV															
RESET	X															
R/W	R/W															
SPR	159															

NOTE: X = "Don't Care".

BARV—Breakpoint Address Register Value

The field represents the value to be compared to the breakpoint.

20.6.2.5 INSTRUCTION SUPPORT CONTROL REGISTER. The instruction support control (ICTRL) register controls instruction operation.

ICTRL

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	CTA			CTB			CTC			CTD			IW0		IW1	
RESET	0			0			0			0			0		0	
R/W	R/W			R/W			R/W			R/W			R/W		R/W	
SPR	158															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	IW2		IW3		SIW0EN	SIW1EN	SIW2EN	SIW3EN	DIW0EN	DIW1EN	DIW2EN	DIW3EN	IFM	ISCT_SER		
RESET	0		0		0	0	0	0	0	0	0	0	0	0		
R/W	R/W		R/W		R/W	R/W	R/W	R/W	R	R	R	R	R/W	R/W		
SPR	158															

CTA—Compare Type of Comparator A

0xx = Not active (reset value).
100 = Equal to.
101 = Less than.
110 = Greater than.
111 = Not equal to.

CTB—Compare Type of Comparator B

0xx = Not active (reset value).
100 = Equal to.
101 = Less than.
110 = Greater than.
111 = Not equal to.

CTC—Compare Type of Comparator C

0xx = Not active (reset value).
100 = Equal to.
101 = Less than.
110 = Greater than.
111 = Not equal to.

CTD—Compare Type of Comparator D

0xx = Not active (reset value).
100 = Equal to.
101 = Less than.
110 = Greater than.
111 = Not equal to.

IW0—Instruction Watchpoint 0

0x = Not active (reset value).
10 = Match from comparator A.
11 = Match from comparators A & B.

IW1—Instruction Watchpoint 1

0x = Not active (reset value).
10 = Match from comparator A.
11 = Match from comparators A | B.

IW2—Instruction Watchpoint 2

0x = Not active (reset value).
10 = Match from comparator C.
11 = Match from comparators C & D.

IW3—Instruction Watchpoint 3

0x = Not active (reset value).
10 = Match from comparator C.
11 = Match from comparators C | D.

SIW0EN—Software Instruction Watchpoint Trap Enable 0

0 = Trap disabled (reset value).
1 = Trap enabled.

SIW1EN—Software Instruction Watchpoint Trap Enable 1

0 = Trap disabled (reset value).
1 = Trap enabled.

SIW2EN—Software Instruction Watchpoint Trap Enable 2

0 = Trap disabled (reset value).
1 = Trap enabled.

SIW3EN—Software Instruction Watchpoint Trap Enable 3

0 = Trap disabled (reset value).
1 = Trap enabled.

DIW0EN—Development Port Instruction Watchpoint Trap Enable 0

This is a read-only bit.

0 = Trap disabled (reset value).
1 = Trap enabled.

DIW1EN—Development Port Instruction Watchpoint Trap Enable 1

This is a read-only bit.

0 = Trap disabled (reset value).
1 = Trap enabled.

DIW2EN—Development Port Instruction Watchpoint Trap Enable 2
This is a read-only bit.

0 = Trap disabled (reset value).
1 = Trap enabled.

DIW3EN—Development Port Instruction Watchpoint Trap Enable 3
This is a read-only bit.

0 = Trap disabled (reset value).
1 = Trap enabled.

IFM—Ignore First Match Only for Instruction Breakpoints
0 = Do not ignore first match. Used for "Go To x" (reset value).
1 = Ignore first match. Used for "Continue".

ISCT_SER—Instruction Fetch Show Cycle and Core Serialize Control
This field defines one of several performance versus visibility options for core behavior. 111 is the preferred encoding for normal operation. Changing the instruction show cycle programming starts to take effect only from the second instruction after the actual **mtspr** instruction to ICTRL.

000 = Core is fully serialized and show cycle will be performed for all fetched instructions (reset value). Has a reset value of 0x00000000.
001 = Core is fully serialized and show cycle will be performed for all changes in the program flow.
010 = Core is fully serialized and show cycle will be performed for all indirect changes in the program flow.
011 = Core is fully serialized and no show cycles will be performed for fetched instructions.
100 = Illegal.
101 = Core is not serialized (normal mode) and show cycle will be performed for all changes in the program flow. If the fetch of the target of a direct branch is aborted by the core, the target is not always visible on the external pins. This does not affect program trace.
110 = Core is not serialized (normal mode) and show cycle will be performed for all indirect changes in the program flow.
111 = Core is not serialized (normal mode) and no show cycles will be performed for fetched instructions.

For more information, see **Section 20.2.2 Controlling Instruction Fetch Show Cycles**.

20.6.2.6 LOAD/STORE SUPPORT COMPARATORS CONTROL REGISTER.

The load/store support comparators control (LCTRL1) register controls the load/store breakpoint and watchpoint operation.

LCTRL1

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	CTE			CTF			CTG			CTH			CRWE		CRWF	
RESET	0			0			0			0			0		0	
R/W	R/W			R/W			R/W			R/W			R/W		R/W	
SPR	156															

BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	CSG		CSH		SUSG	SUSH	CGBMSK				CHBMSK				RESERVED	
RESET	0		0		0	0	0				0				0	
R/W	R/W		R/W		R/W	R/W	R/W				R/W				R/W	
SPR	156															

CTE—Compare Type, Comparator E

0xx = Not active (reset value).
100 = Equal to.
101 = Less than.
110 = Greater than.
111 = Not equal to.

CTF—Compare Type, Comparator F

0xx = Not active (reset value).
100 = Equal to.
101 = Less than.
110 = Greater than.
111 = Not equal to.

CTG—Compare Type, Comparator G

0xx = Not active (reset value).
100 = Equal to.
101 = Less than.
110 = Greater than.
111 = Not equal to.

CTH—Compare Type, Comparator H

0xx = Not active (reset value).
100 = Equal to.
101 = Less than.
110 = Greater than.
111 = Not equal to.

CRWE—Select Match on Read/Write of Comparator E

0x = "Don't care" (reset value).
10 = Match on read.
11 = Match on write.

CRWF—Select Match on Read/Write of Comparator F

0x = "Don't care" (reset value).
10 = Match on read.
11 = Match on write.

CSG—Compare Size, Comparator G

00 = Reserved.
01 = Word.
10 = Half-word.
11 = Byte.

CSH—Compare Size, Comparator H

00 = Reserved.
01 = Word.
10 = Half-word.
11 = Byte.

SUSG—Signed/Unsigned Operating Mode for Comparator G

0 = Signed.
1 = Unsigned.

SUSH—Signed/Unsigned Operating Mode for Comparator H

0 = Signed.
1 = Unsigned.

CGBMSK—Byte Mask for Comparator G

0000 = All bytes are not masked.
0001 = Last byte of the word is masked.
•
•
•
1111 =All bytes are masked.

CHBMSK—Byte Mask for Comparator H

0000 = All bytes are not masked.
0001 = Last byte of the word is masked.
•
•
•
1111 = All bytes are masked.

Bits 30–31—Reserved

These bits are reserved and should be set to 0.

20.6.2.7 LOAD/STORE SUPPORT AND-OR CONTROL REGISTER. The load/store support AND-OR control (LCTRL2) register is used to control the bit masks for load/store data comparisons. Watchpoint programming consists of three control register fields—LWxIA, LWxLA, and LWxLD. All three conditions must be detected to assert a watchpoint. The reset value of this register is 0x00000000.

LCTRL2

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	LW0EN	LW0IA		LW0IADC	LW0LA		LW0LADC	LW0LD		LW0LDDC	LW1EN	LW1IA		LW1IADC	LW1LA	
RESET	0	0		0	0		0	0		0	0	0		0	0	
R/W	R/W	R/W		R/W	R/W		R/W	R/W		R/W	R/W	R/W		R/W	R/W	
SPR	157															

BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	LW1LADC	LW1LD		LW1LDDC	BRKNOMSK	RESERVED							DLW0EN	DLW1EN	SLW0EN	SLW1EN
RESET	0	0		0	0	0							0	0	0	0
R/W	R/W	R/W		R/W	R/W	R/W							R/W	R/W	R/W	R/W
SPR	157															

LW0EN—First Load/Store Watchpoint Enable

0 = Watchpoint not enabled (reset value).
1 = Watchpoint enabled.

LW0IA—First Load/Store Watchpoint I-Address Watchpoint Selection

00 = First instruction watchpoint.
01 = Second instruction watchpoint.
10 = Third instruction watchpoint.
11 = Fourth instruction watchpoint.

LW0IADC—First Load/Store Watchpoint Care/Don't Care I-Address Events

0 = "Don't care."
1 = "Care."

LW0LA—First Load/Store Watchpoint L-Address Events Selection

00 = Match from comparator E.
01 = Match from comparator F.
10 = Match from comparators E & F.
11 = Match from comparators E | F.

LW0LADC—First Load/Store Watchpoint Care/Don't Care L-Address Events

0 = "Don't care."
1 = "Care."

LW0LD—First Load/Store Watchpoint L-Data Events Selection

00 = Match from comparator G.
01 = Match from comparator H.
10 = Match from comparators G & H.
11 = Match from comparators G | H.

LW0LDDC—First Load/Store Watchpoint Care/Don't Care L-Data Events

0 = "Don't care."
1 = "Care."

LW1EN—Second Load/Store Watchpoint Enable

0 = Watchpoint not enabled (reset value).
1 = Watchpoint enabled.

LW1IA—Second Load/Store Watchpoint I-Address Watchpoint Selection

00 = First instruction watchpoint.
01 = Second instruction watchpoint.
10 = Third instruction watchpoint.
11 = Fourth instruction watchpoint.

LW1IADC—Second Load/Store Watchpoint Care/Don't Care I-Address Event

0 = "Don't care."
1 = "Care."

LW1LA—Second Load/Store Watchpoint L-Address Events Selection

00 = Match from comparator E.
01 = Match from comparator F.
10 = Match from comparators E & F.
11 = Match from comparators E | F.

LW1LADC—Second Load/Store Watchpoint Care/Don't Care L-Address Events

0 = "Don't care."
1 = "Care."

LW1LD—Second Load/Store Watchpoint L-Data Events Selection

00 = Match from comparator G.
01 = Match from comparator H.
10 = Match from comparators G & H.
11 = Match from comparator G | H.

LW1LDDC—Second Load/Store Watchpoint Care/Don't Care L-Data Events

0 = "Don't care."
1 = "Care."

BRKNOMSK—Internal Breakpoints Nonmask Bit Controls Both Instruction Breakpoints and Load/Store Breakpoints

0 = Masked mode, breakpoints are recognized only when MSR_{RI} =1 (reset value).
1 = Nonmasked mode, breakpoints are always recognized.

Bits 21–27—Reserved

These bits are reserved and should be set to 0.

DLW0EN—Development Port Trap Enable Selection of the First Load/Store Watchpoint (Read-Only Bit)

0 = Trap disabled (reset value).
1 = Trap enabled.

DLW1EN—Development Port Trap Enable Selection of the Second Load/Store Watchpoint (Read-Only Bit)

0 = Trap disabled (reset value).
1 = Trap enabled.

SLW0EN—Software Trap Enable Selection of the First Load/Store Watchpoint

0 = Trap disabled (reset value).
1 = Trap enabled.

SLW1EN—Software Trap Enable Selection of the Second Load/Store Watchpoint

0 = Trap disabled (reset value).
1 = Trap enabled.

20.6.2.8 BREAKPOINT COUNTER A VALUE AND CONTROL REGISTER. The breakpoint counter A value and control (COUNTA) register is used to count watchpoint events and to generate breakpoints a programmable amount of time after a watchpoint occurs.

COUNTA

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	CNTV															
RESET	—															
R/W	R/W															
SPR	150															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED														CNTC	
RESET	0														0	
R/W	R/W														R/W	
SPR	150															

NOTE: — = Undefined.

CNTV—Counter Preset Value

This field contains the number of watchpoint events to be encountered before a breakpoint is generated.

Bits 16–29—Reserved

These bits are reserved and should be set to 0.

CNTC—Counter Source Select

00 = Not active (reset value).
01 = Instruction first watchpoint.
10 = Load/store first watchpoint.
11 = Reserved.

20.6.2.9 BREAKPOINT COUNTER B VALUE AND CONTROL REGISTER. The breakpoint counter B value and control (COUNTB) register is used to count watchpoint events and to generate breakpoints a programmable amount of time after a watchpoint occurs.

COUNTB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	CNTV															
RESET	—															
R/W	R/W															
SPR	151															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RESERVED														CNTC	
RESET	—														—	
R/W	R/W														R/W	
SPR	151															

NOTE: — = Undefined.

CNTV—Counter Preset Value

This field contains the number of watchpoint events to be encountered before a breakpoint is generated.

Bits 16–29—Reserved

These bits are reserved and should be set to 0.

CNTC—Counter Source Select

00 = Not active (reset value).
01 = Instruction second watchpoint.
10 = Load/store second watchpoint.
11 = Reserved.

20.6.3 Debug Mode Registers

20.6.3.1 INTERRUPT CAUSE REGISTER. The interrupt cause register (ICR) provides the reason for entering debug mode. All bits are set by the hardware, cleared when the register is read, and cleared to zero when exiting reset. Any attempt to write to this register is ignored. The reset value for this register is 0x00000000.

ICR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RES	RST	CHSTP	MCI	RESERVED		EXTI	ALI	PRI	FPUVI	DECI	RESERVED		SYSI	TR	RES
RESET	0	0	0	0	0		0	0	0	0	0	0		0	0	0
R/W	R	R	R	R	R		R	R	R	R	R	R		R	R	R
SPR	148															

BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RES	SEI	ITLBMS	DTLBMS	ITLBER	DTLBER	RESERVED						LBRK	IBRK	EBRK	DPI
RESET	0	0	0	0	0	0	0						0	0	0	0
R/W	R	R	R	R	R	R	R						R	R	R	R
SPR	148															

Bits 0, 4, and 5—Reserved

These bits are reserved and should be set to 0.

RST—Reset Interrupt

This bit is set when the system reset pin is asserted. This pin is not implemented in the core.

CHSTP—Check Stop

This bit is set when the machine check interrupt is asserted and MSR_{ME} =0. The core enters debug mode if enabled and the CHSTP bit in the DER is set. Otherwise, the processor enters the check stop state.

MCI—Machine Check Interrupt

This bit is set when the machine check interrupt is asserted and MSR_{ME} =1. The core enters debug mode if enabled and the MCI bit in the DER is set.

EXTI—External Interrupt

This bit is set when the external interrupt is asserted. The core enters debug mode if enabled and the EXTI bit in the DIR is set.

ALI—Alignment Interrupt

This bit is set when the alignment interrupt is asserted. The core enters debug mode if enabled and the ALI bit in the DIR is set.

PRI—Program Interrupt

This bit is set when the program interrupt is asserted.The core enters in debug mode if enabled and the PRI bit in the DER is set.

FPUVI—Floating-Point Unavailable Interrupt

This bit is set when the floating-point unavailable interrupt is asserted. The core enters debug mode if enabled and the FPUVI bit in the DER is set.

DECI—Decrementer Interrupt

This bit is set when the decrementer interrupt is asserted. The core enters debug mode if enabled and the DECI bit in the DER is set.

Bits 11–12, 15–16—Reserved

These bits are reserved and should be set to 0.

SYSI—System Call Interrupt

This bit is set when the system call interrupt is asserted. The core enters debug mode if enabled and the SYSI bit in the DER is set.

TR—Trace Interrupt

This bit is set when in single-step mode or when in branch trace mode. The core enters debug mode if enabled and the TR bit in the DER is set.

SEI—Implementation Dependent Software Emulation Interrupt

This bit is set when the floating-point assist interrupt is asserted. The core enters debug mode if enabled and the SEI bit in the DER is set.

ITLBMS—Implementation Specific Instruction TLB Miss

This bit is set as a result of an instruction TLB miss.The core enters debug mode if enabled and the ITLBMS bit in the DER is set.

DTLBMS—Implementation Specific Data TLB Miss

This bit is set as a result of an data TLB miss. The core enters debug mode if enabled and the DTLBMS bit in the DER is set.

ITLBER—Implementation Specific Instruction TLB Error

This bit is set as a result of an instruction TLB error. The core enters debug mode if enabled and the ITLBER bit in the DER is set.

DTLBER—Implementation Specific Data TLB Error

This bit is set as a result of an data TLB error. The core enters debug mode if enabled and the DTLBER bit in the DER is set.

Bits 22–27—Reserved

These bits are reserved and should be set to 0.

LBRK—Load/Store Breakpoint Interrupt

This bit is set as a result of the assertion of an load/store breakpoint. The core enters debug mode if enabled and the LBRK bit in the DER is set.

IBRK—Instruction Breakpoint Interrupt

This bit is set as a result of the assertion of an instruction breakpoint. The core enters debug mode if enabled and the IBRK bit in the DER is set.

EBRK—External Breakpoint Interrupt

This bit is set as a result of the assertion of an external breakpoint. The core enters debug mode if enabled and the EBRK bit in the DER is set.

DPI—Development Port Interrupt

This bit is set by the development port as a result of a debug station nonmaskable request or when entering debug mode immediately out of reset. The core enters debug mode if enabled and the DPI bit in the DER is set.

20.6.3.2 DEBUG ENABLE REGISTER. The debug enable register (DER) allows the enabling of events that cause the processor to enter debug mode.

DER

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RES	RSTE	CHSTPE	MCIE	RESERVED		EXTIE	ALIE	PRIE	FPUVIE	DECIE	RESERVED		SYSIE	TRE	RES
RESET	0	0	1	0	0		0	0	0	0	0	0		0	1	0
R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W
SPR	149															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	RES	SEIE	ITLBMSE	DTLBMSE	ITLBERE	DTLBERE	RESERVED						LBRKE	IBRKE	EBRKE	DPIE
RESET	0	0	0	0	0	0	0						1	1	1	1
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W						R/W	R/W	R/W	R/W
SPR	149															

Bits 0, 4, and 5—Reserved

These bits are reserved and should be set to 0.

RSTE—Reset Interrupt Enable

0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

CHSTPE—Check Stop Enable
0 = Debug mode entry is disabled.
1 = Debug mode entry is enabled (reset value).

MCIE—Machine Check Interrupt Enable
0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

EXTIE—External Interrupt Enable
0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

ALIE—Alignment Interrupt Enable
0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

PRIE—Program Interrupt Enable
0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

FPUVIE—Floating-Point Unavailable Interrupt Enable
0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

DECIE—Decrementer Interrupt Enable
0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

Bits 11–12 and 15–16—Reserved
These bits are reserved and should be set to 0.

SYSIE—System Call Interrupt Enable
0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

TRE—Trace Interrupt Enable
0 = Debug mode entry is disabled.
1 = Debug mode entry is enabled (reset value).

SEIE—Software Emulation Interrupt Enable
0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

ITLBMSE—Implementation Specific Instruction TLB Miss Enable
0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

ITLBERE—Implementation Specific Instruction TLB Error Enable

0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

DTLBMSE—Implementation Specific Data TLB Miss Enable

0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

DTLBERE—Implementation Specific Data TLB Error Enable

0 = Debug mode entry is disabled (reset value).
1 = Debug mode entry is enabled.

Bits 22–27—Reserved

These bits are reserved and should be set to 0.

LBRKE—Load/Store Breakpoint Interrupt Enable

0 = Debug mode entry is disabled.
1 = Debug mode entry is enabled (reset value).

IBRKE—Instruction Breakpoint Interrupt Enable

0 = Debug mode entry is disabled.
1 = Debug mode entry is enabled (reset value).

EBRKE—External Breakpoint Interrupt Enable

0 = Debug mode entry is disabled.
1 = Debug mode entry is enabled (reset value).

DPIE—Development Port Nonmaskable Request Enable

0 = Debug mode entry is disabled.
1 = Debug mode entry is enabled (reset value).

20.6.4 Development Port Data Register

The special-purpose development port data register (DPDR) physically resides in the development port logic. It is used for data interchange between the core and development system. An access to this register is initiated using the **mtspr** and **mfspr** instructions and it is implemented using a special bus cycle on the internal bus. For details, see Table 6-9.

DPDR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	DPDV															
RESET	—															
R/W	R/W															
SPR	630															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	DPDV															
RESET	—															
R/W	R/W															
SPR	630															

NOTE: — = Undefined.

DPDV—Development Port Data Value

This field contains the data to be transferred between the core and the development system.

DPIR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	DPIV															
RESET	—															
R/W	R															
SPR	631															
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	DPIV															
RESET	—															
R/W	R															
SPR	631															

NOTE: — = Undefined.

DPIV—Development Port Instruction Value

This field is used to fetch an instruction supplied by the development system.

SECTION 21
IEEE 1149.1 TEST ACCESS PORT

The MPC823 provides a dedicated user-accessible test access port (TAP) that is fully compatible with the *IEEE 1149.1 Standard Test Access Port and Boundary Scan Architecture*. Problems associated with testing high-density circuit boards have led tothe development of this proposed standard under the sponsorship of the Test Technology Committee of IEEE and the Joint Test Action Group (JTAG). The MPC823 implementation supports circuit board test strategies based on this standard.

The TAP consists of five dedicated signal pins, a 16-state TAP controller, and two test data registers. A boundary scan register links all the device signal pins into a single shift register. The test logic, which is implemented using static logic design, is independent of the device system logic. The MPC823 implementation provides the capability to:

- Perform boundary scan operations to check circuit board electrical continuity.
- Bypass the MPC823 for a given circuit board test by effectively reducing the boundary scan register to a single cell.
- Sample the MPC823 system pins during operation and transparently shift out the result in the boundary scan register.
- Disable the output drive to pins during circuit board testing.

Note: Certain precautions must be observed to ensure that the IEEE 1149.1-like test logic does not interfere with nontest operation.

The MPC823 implementation includes a TAP controller, a 4-bit instruction register, and two test registers (a 1-bit bypass register and a 397-bit boundary scan register). An overview of the MPC823 scan chain implementation is illustrated in the figure below. The TAP controller consists of the following signals:

- TCK—A test clock input to synchronize the test logic.
- TMS—A test mode select input (with an internal pull-up resistor) that is sampled on the rising edge of TCK to sequence the TAP controller's state machine.
- TDI—A test data input (with an internal pull-up resistor) that is sampled on the rising edge of TCK.
- TDO—A three-stateable test data output that is actively driven in the shift-IR and shift-DR controller states. TDO changes on the falling edge of TCK.
- $\overline{\text{TRST}}$—An asynchronous reset with an internal pull-up resistor that provides TAP controller initialization and other logic required by the standard. For normal operation of the MPC823, this signal pin must make a level transition to low before initialization begins. Typically, if the TAP is used, connect the $\overline{\text{TRST}}$ signal to $\overline{\text{PORESET}}$, $\overline{\text{HRESET}}$, or $\overline{\text{SRESET}}$.

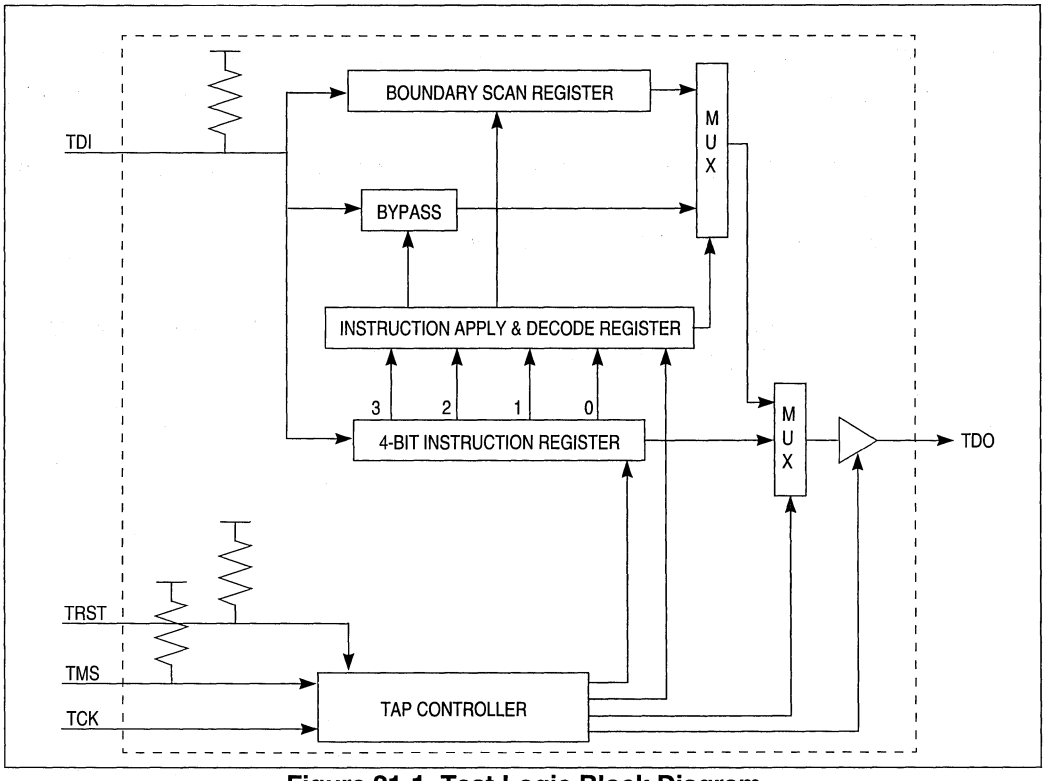


Figure 21-1. Test Logic Block Diagram

21.1 THE TAP CONTROLLER

The TAP controller is responsible for interpreting the sequence of logical values on the TMS signal. It is a synchronous state machine that controls the operation of the JTAG logic. The value shown adjacent to each bubble in the figure below represents the value of the TMS signal sampled on the rising edge of the TCK signal.

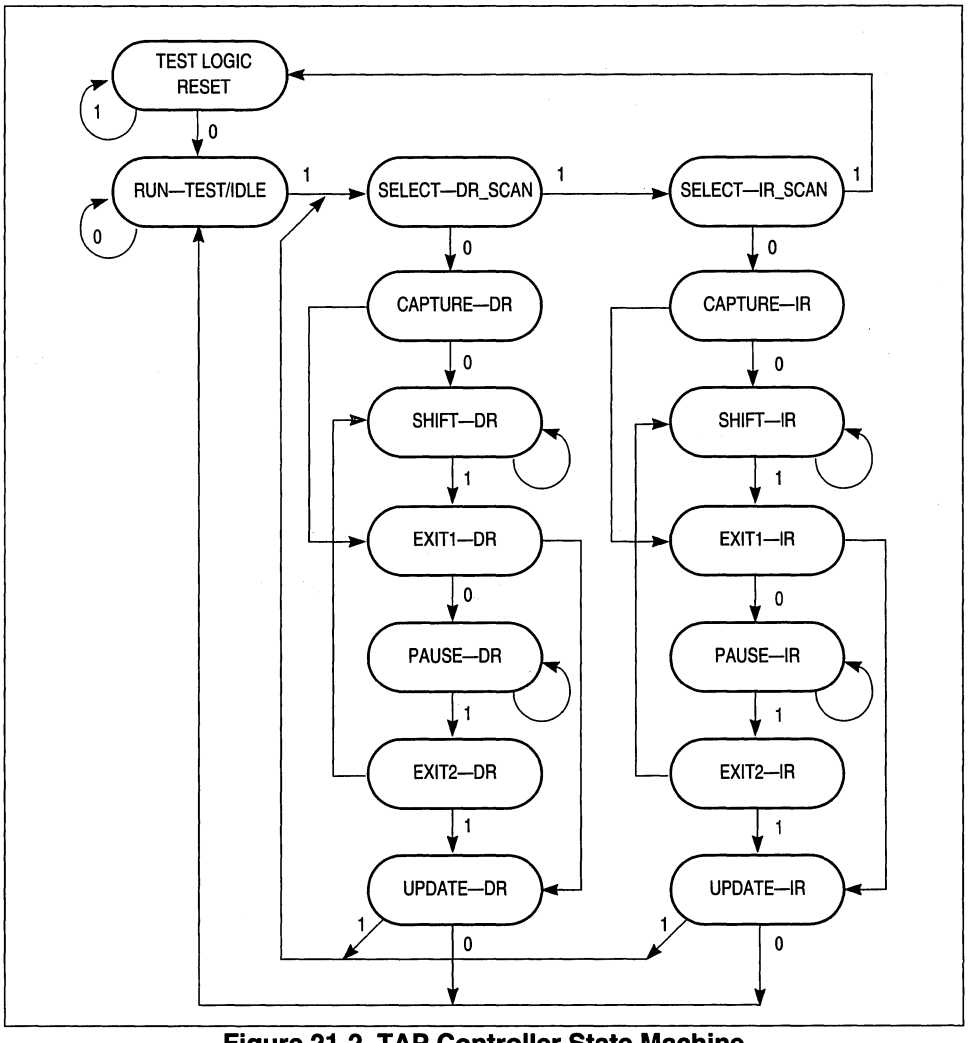


Figure 21-2. TAP Controller State Machine

21.2 THE BOUNDARY SCAN REGISTER

The MPC823 scan chain implementation has a 397-bit boundary scan register that contains bits for all device signal, clock pins, and the associated control signals. However, the XTAL, EXTAL, and XFC pins are associated with analog signals and are not included in the boundary scan register. An IEEE-1149.1-compliant boundary scan register has been included on the MPC823. This 397-bit boundary scan register can be connected between the TDI and TDO signals when the **extest** or **sample/preload** instructions are selected. It is used for capturing signal pin data on the input pins, forcing fixed values on the output signal pins, and selecting the direction and drive characteristics (a logic value or high impedance) of the bidirectional and three-state signal pins. Figure 21-3 through Figure 21-6 depict the various cell types.

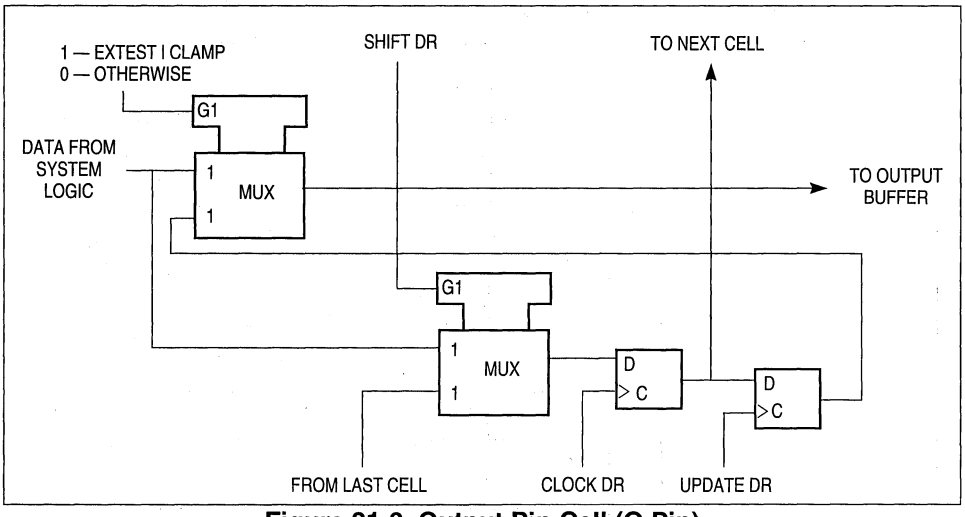


Figure 21-3. Output Pin Cell (O.Pin)

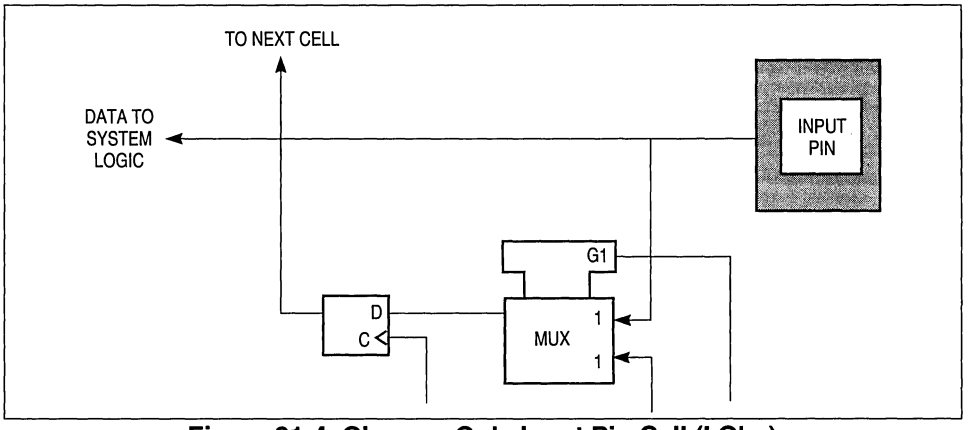


Figure 21-4. Observe-Only Input Pin Cell (I.Obs)

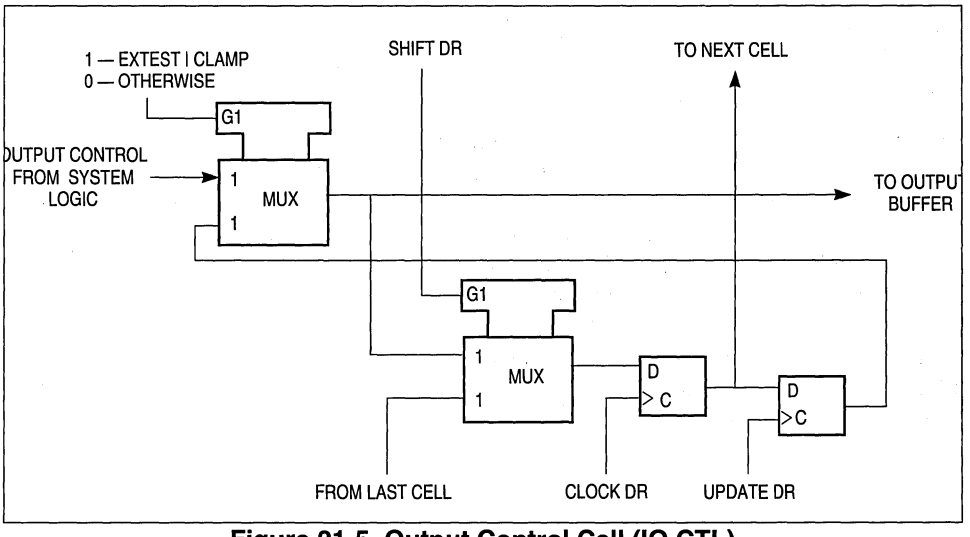


Figure 21-5. Output Control Cell (IO.CTL)

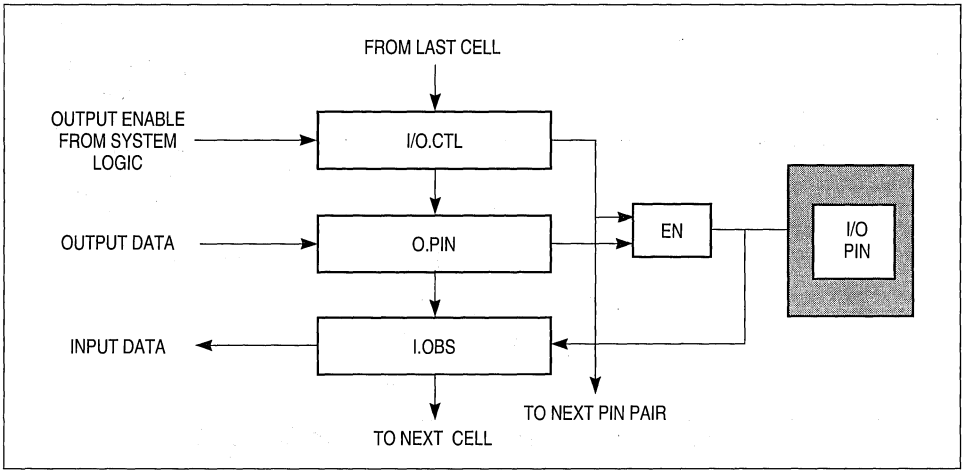


Figure 21-6. General Arrangement of Bidirectional Pin Cells

The value of the control bit controls the output function of the bidirectional pin. One or more bidirectional data cells can be serially connected to a control cell. Bidirectional pins include two scan cell for data (IO.Cell) as illustrated in Figure 21-6 and these bits are controlled by the cell illustrated in Figure 21-5.

It is important to know the boundary scan bit order and the pins that are associated with them. The bit order starting with the TDO output and ending with the TDI input is shown in Table 21-1. The first column of the table defines the bit's ordinal position in the boundary scan register. The shift register cell nearest TDO (first to be shifted in) is defined as Bit 0 and the last bit to be shifted in is Bit 396. The second column references one of the three MPC823 cell types depicted in Figure 21-3 through Figure 21-6 that describe the cell structure for each type. The third column lists the pin name for all pin-related cells and defines the name of the bidirectional control register bits. The fourth column lists the pin type and the last column indicates the associated boundary scan register control bit for the bidirectional output pins.

Table 21-1. Boundary Scan Bit Definition

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
1	i.obs	PB[26]	IO	—
2	o.pin	PB[26]	IO	g56.ctl
3	IO.ctl	G56.CTL	—	—
4	i.obs	PC[12]	IO	—
5	o.pin	PC[12]	IO	g34.ctl
6	IO.ctl	G34.CTL	—	—
7	i.obs	PA[12]	IO	—
8	o.pin	PA[12]	IO	g74.ctl
9	IO.ctl	G74.CTL	—	—
10	i.obs	PB[27]	IO	—
11	o.pin	PB[27]	IO	g57.ctl
12	IO.ctl	G57.CTL	—	—
13	i.obs	PC[13]	IO	—
14	o.pin	PC[13]	IO	g35.ctl
15	IO.ctl	G35.CTL	—	—
16	i.obs	PA[13]	IO	—
17	o.pin	PA[13]	IO	g75.ctl
18	IO.ctl	G75.CTL	—	—
19	i.obs	PB[28]	IO	—
20	o.pin	PB[28]	IO	g58.ctl
21	IO.ctl	G58.CTL	—	—
22	i.obs	PC[14]	IO	—
23	o.pin	PC[14]	IO	g36.ctl
24	IO.ctl	G36.CTL	—	—
25	i.obs	PA[14]	IO	—
26	o.pin	PA[14]	IO	g76.ctl
27	IO.ctl	G76.CTL	—	—
28	i.obs	PB[29]	IO	—
29	o.pin	PB[29]	IO	g59.ctl
30	IO.ctl	G59.CTL	—	—
31	i.obs	PC[15]	IO	—
32	o.pin	PC[15]	IO	g37.ctl

Table 21-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
33	IO.ctl	G37.CTL	—	—
34	i.obs	PB[30]	IO	—
35	o.pin	PB[30]	IO	g60.ctl
36	IO.ctl	G60.CTL	—	—
37	i.obs	PA[15]	IO	—
38	o.pin	PA[15]	IO	g77.ctl
39	IO.ctl	G77.CTL	—	—
40	i.obs	PB[31]	IO	—
41	o.pin	PB[31]	IO	g61.ctl
42	IO.ctl	G61.CTL	—	—
43	i.obs	A[6]	IO	—
44	o.pin	A[6]	IO	g203.ctl
45	IO.ctl	G203.CTL	—	—
46	i.obs	A[7]	IO	—
47	o.pin	A[7]	IO	g203.ctl
48	i.obs	A[8]	IO	—
49	o.pin	A[8]	IO	g202.ctl
50	IO.ctl	G202.CTL	—	—
51	i.obs	A[9]	IO	—
52	o.pin	A[9]	IO	g202.ctl
53	i.obs	A[10]	IO	—
54	o.pin	A[10]	IO	g202.ctl
55	i.obs	A[11]	IO	—
56	o.pin	A[11]	IO	g202.ctl
57	i.obs	A[12]	IO	—
58	o.pin	A[12]	IO	g202.ctl
59	i.obs	A[13]	IO	—
60	o.pin	A[13]	IO	g202.ctl
61	i.obs	A[14]	IO	—
62	o.pin	A[14]	IO	g202.ctl
63	i.obs	A[15]	IO	—
64	o.pin	A[15]	IO	g202.ctl
65	i.obs	A[16]	IO	—
66	o.pin	A[16]	IO	g201.ctl

Table 21-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
67	IO.ctl	G201.CTL	—	—
68	i.obs	A[17]	IO	—
69	o.pin	A[17]	IO	g201.ctl
70	i.obs	A[19]	IO	—
71	o.pin	A[19]	IO	g201.ctl
72	i.obs	A[27]	IO	—
73	o.pin	A[27]	IO	g201.ctl
74	i.obs	A[20]	IO	—
75	o.pin	A[20]	IO	g201.ctl
76	i.obs	A[21]	IO	—
77	o.pin	A[21]	IO	g201.ctl
78	i.obs	A[24]	IO	—
79	o.pin	A[24]	IO	g201.ctl
80	i.obs	A[23]	IO	—
81	o.pin	A[23]	IO	g201.ctl
82	i.obs	A[29]	IO	—
83	o.pin	A[29]	IO	g200.ctl
84	IO.ctl	G200.CTL	—	—
85	i.obs	A[25]	IO	—
86	o.pin	A[25]	IO	g200.ctl
87	i.obs	A[30]	IO	—
88	o.pin	A[30]	IO	g200.ctl
89	i.obs	A[18]	IO	—
90	o.pin	A[18]	IO	g200.ctl
91	i.obs	A[28]	IO	—
92	o.pin	A[28]	IO	g200.ctl
93	i.obs	A[22]	IO	—
94	o.pin	A[22]	IO	g200.ctl
95	i.obs	A[26]	IO	—
96	o.pin	A[26]	IO	g200.ctl
97	i.obs	A[31]	IO	—
98	o.pin	A[31]	IO	g200.ctl
99	i.obs	TSIZ0_REG_B	IO	—
100	o.pin	TSIZ0_REG_B	IO	g204.ctl

Table 21-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
101	i.obs	TSIZ1	IO	—
102	o.pin	TSIZ1	IO	g204.ctl
103	IO.ctl	G204.CTL	—	—
104	o.pin	WE3_B_BSAB3_B_PCWEB	O	—
105	o.pin	WE1_B_BSAB1_B_IOWR_B	O	—
106	o.pin	WE2_B_BSAB2_B_PCOE_B	O	—
107	o.pin	WE0_B_BSAB0_B_IORD_B	O	—
108	o.pin	GPLA0_B_GPLB0_B	O	—
109	o.pin	OE_B_GPLAB1_B	O	—
110	o.pin	GPLAB2_B_CS2_B	O	—
111	o.pin	GPLAB3_B_CS3_B	O	—
112	o.pin	CS4_B	O	—
113	o.pin	CS5_B	O	—
114	o.pin	CS6_B_CE1B_B	O	—
115	o.pin	CS7_B_CE2B_B	O	—
116	o.pin	CS3_B	O	—
117	o.pin	CS2_B	O	—
118	o.pin	CS1_B	O	—
119	o.pin	CS0_B	O	—
120	i.obs	WR_B	IO	—
121	o.pin	WR_B	IO	g96.ctl
122	IO.ctl	G96.CTL	—	—
123	i.obs	GPLB4_B_UPWAITB	IO	—
124	o.pin	GPLB4_B_UPWAITB	IO	g24.ctl
125	IO.ctl	G24.CTL	—	—
126	o.pin	GPLA5_B	O	—
127	i.obs	GPLA4_B_UPWAITA	IO	—
128	o.pin	GPLA4_B_UPWAITA	IO	g25.ctl
129	IO.ctl	G25.CTL	—	—
130	o.pin	BDIP_B_GPLB5_B	O	—
131	i.obs	BI_B	IO	—
132	o.pin	BI_B	IO	g23.ctl
133	IO.ctl	G23.CTL	—	—
134	i.obs	TA_B	IO	—

Table 21-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
135	o.pin	TA_B	IO	g22.ctl
136	IO.ctl	G22.CTL	—	—
137	i.obs	TEA_B	IO	—
138	o.pin	TEA_B	IO	g89.ctl
139	IO.ctl	G89.CTL	—	—
140	i.obs	TS_B	IO	—
141	o.pin	TS_B	IO	g97.ctl
142	IO.ctl	G97.CTL	—	—
143	i.obs	BR_B	IO	—
144	o.pin	BR_B	IO	g21.ctl
145	IO.ctl	G21.CTL	—	—
146	i.obs	BG_B	IO	—
147	o.pin	BG_B	IO	g20.ctl
148	IO.ctl	G20.CTL	—	—
149	i.obs	BB_B	IO	—
150	o.pin	BB_B	IO	g11.ctl
151	IO.ctl	G11.CTL	—	—
152	i.obs	FRZ_IRQ6_B	IO	—
153	o.pin	FRZ_IRQ6_B	IO	g90.ctl
154	IO.ctl	G90.CTL	—	—
155	i.obs	BURST_B	IO	—
156	o.pin	BURST_B	IO	g205.ctl
157	IO.ctl	G205.CTL	—	—
158	i.obs	RSV_B_IRQ2_B	IO	—
159	o.pin	RSV_B_IRQ2_B	IO	g17.ctl
160	IO.ctl	G17.CTL	—	—
161	i.obs	IPB5_LWP1_VF1	IO	—
162	o.pin	IPB5_LWP1_VF1	IO	g15.ctl
163	IO.ctl	G15.CTL	—	—
164	i.obs	IPB4_LWP0_VF0	IO	—
165	o.pin	IPB4_LWP0_VF0	IO	g150.ctl
166	IO.ctl	G150.CTL	—	—
167	i.obs	IPB3_IWP2_VF2	IO	—
168	o.pin	IPB3_IWP2_VF2	IO	g151.ctl

Table 21-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
169	IO.ctl	G151.CTL	—	—
170	i.obs	IPB1_IWP1_VFLS1	IO	—
171	o.pin	IPB1_IWP1_VFLS1	IO	g150.ctl
172	i.obs	IPB0_IWP0_VFLS0	IO	—
173	o.pin	IPB0_IWP0_VFLS0	IO	g150.ctl
174	i.obs	IPB7_PTR_AT3	IO	—
175	o.pin	IPB7_PTR_AT3	IO	g13.ctl
176	IO.ctl	G13.CTL	—	—
177	i.obs	IPB2_IOIS16B_B_AT2	IO	—
178	o.pin	IPB2_IOIS16B_B_AT2	IO	g41.ctl
179	IO.ctl	G41.CTL	—	—
180	i.obs	ALEB_DSCK_AT1	IO	—
181	o.pin	ALEB_DSCK_AT1	IO	g16.ctl
182	IO.ctl	G16.CTL	—	—
183	i.obs	IPB6_DSDI_AT0	IO	—
184	o.pin	IPB6_DSDI_AT0	IO	g14.ctl
185	IO.ctl	G14.CTL	—	—
186	i.obs	KR_B_IRQ4_B_SPKROUT	IO	—
187	o.pin	KR_B_IRQ4_B_SPKROUT	IO	g95.ctl
188	IO.ctl	G95.CTL	—	—
189	i.obs	OP2_MODCK1_STS_B	IO	—
190	o.pin	OP2_MODCK1_STS_B	IO	g92.ctl
191	IO.ctl	G92.CTL	—	—
192	i.obs	OP3_MODCK2_DSDO	IO	—
193	o.pin	OP3_MODCK2_DSDO	IO	g91.ctl
194	IO.ctl	G91.CTL	—	—
195	i.obs	CLK4IN	I	—
196	o.pin	TEXP	O	—
197	i.obs	HRESET_B	IO	—
198	o.pin	HRESET_B	IO	g94.ctl
199	IO.ctl	G94.CTL	—	—
200	i.obs	SRESET_B	IO	—
201	o.pin	SRESET_B	IO	g93.ctl
202	IO.ctl	G93.CTL	—	—

Table 21-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
203	i.obs	RSTCONF_B	I	—
204	i.obs	PORESET_B	I	—
205	i.obs	WAITB_B	I	—
206	o.pin	CLKOUT	O	—
207	i.obs	DP0_IRQ3_B	IO	—
208	o.pin	DP0_IRQ3_B	IO	g18.ctl
209	i.obs	DP3_IRQ6_B	IO	—
210	o.pin	DP3_IRQ6_B	IO	g18.ctl
211	IO.ctl	G18.CTL	—	—
212	i.obs	DP2_IRQ5_B	IO	—
213	o.pin	DP2_IRQ5_B	IO	g18.ctl
214	i.obs	DP1_IRQ4_B	IO	—
215	o.pin	DP1_IRQ4_B	IO	g18.ctl
216	i.obs	D[31]	IO	—
217	o.pin	D[31]	IO	g103.ctl
218	IO.ctl	G103.CTL	—	—
219	i.obs	D[30]	IO	—
220	o.pin	D[30]	IO	g103.ctl
221	i.obs	D[29]	IO	—
222	o.pin	D[29]	IO	g103.ctl
223	i.obs	D[7]	IO	—
224	o.pin	D[7]	IO	g103.ctl
225	i.obs	D[28]	IO	—
226	o.pin	D[28]	IO	g103.ctl
227	i.obs	D[26]	IO	—
228	o.pin	D[26]	IO	g103.ctl
229	i.obs	D[25]	IO	—
230	o.pin	D[25]	IO	g103.ctl
231	i.obs	D[24]	IO	—
232	o.pin	D[24]	IO	g103.ctl
233	i.obs	D[21]	IO	—
234	o.pin	D[21]	IO	g102.ctl
235	IO.ctl	G102.CTL	—	—
236	i.obs	D[22]	IO	—

Table 21-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
237	o.pin	D[22]	IO	g102.ctl
238	i.obs	D[6]	IO	—
239	o.pin	D[6]	IO	g102.ctl
240	i.obs	D[20]	IO	—
241	o.pin	D[20]	IO	g102.ctl
242	i.obs	D[19]	IO	—
243	o.pin	D[19]	IO	g102.ctl
244	i.obs	D[18]	IO	—
245	o.pin	D[18]	IO	g102.ctl
246	i.obs	D[15]	IO	—
247	o.pin	D[15]	IO	g102.ctl
248	i.obs	D[16]	IO	—
249	o.pin	D[16]	IO	g102.ctl
250	i.obs	D[5]	IO	—
251	o.pin	D[5]	IO	g101.ctl
252	IO.ctl	G101.CTL	—	—
253	i.obs	D[3]	IO	—
254	o.pin	D[3]	IO	g101.ctl
255	i.obs	D[14]	IO	—
256	o.pin	D[14]	IO	g101.ctl
257	i.obs	D[2]	IO	—
258	o.pin	D[2]	IO	g101.ctl
259	i.obs	D[11]	IO	—
260	o.pin	D[11]	IO	g101.ctl
261	i.obs	D[10]	IO	—
262	o.pin	D[10]	IO	g101.ctl
263	i.obs	D[9]	IO	—
264	o.pin	D[9]	IO	g101.ctl
265	i.obs	D[17]	IO	—
266	o.pin	D[17]	IO	g101.ctl
267	i.obs	D[27]	IO	—
268	o.pin	D[27]	IO	g100.ctl
269	IO.ctl	G100.CTL	—	—
270	i.obs	D[23]	IO	—

Table 21-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
271	o.pin	D[23]	IO	g100.ctl
272	i.obs	D[1]	IO	—
273	o.pin	D[1]	IO	g100.ctl
274	i.obs	D[4]	IO	—
275	o.pin	D[4]	IO	g100.ctl
276	i.obs	D[0]	IO	—
277	o.pin	D[0]	IO	g100.ctl
278	i.obs	D[12]	IO	—
279	o.pin	D[12]	IO	g100.ctl
280	i.obs	D[8]	IO	—
281	o.pin	D[8]	IO	g100.ctl
282	i.obs	D[13]	IO	—
283	o.pin	D[13]	IO	g100.ctl
284	i.obs	IRQ0_B	I	—
285	i.obs	IRQ1_B	I	—
286	i.obs	IRQ7_B	I	—
287	i.obs	SPARE3	IO	—
288	o.pin	SPARE3	IO	g10.ctl
289	IO.ctl	G10.CTL	—	—
290	i.obs	PD[3]	IO	—
291	o.pin	PD[3]	IO	g9.ctl
292	IO.ctl	G9.CTL	—	—
293	i.obs	PD[5]	IO	—
294	o.pin	PD[5]	IO	g8.ctl
295	IO.ctl	G8.CTL	—	—
296	i.obs	PD[6]	IO	—
297	o.pin	PD[6]	IO	g7.ctl
298	IO.ctl	G7.CTL	—	—
299	i.obs	PD[4]	IO	—
300	o.pin	PD[4]	IO	g6.ctl
301	IO.ctl	G6.CTL	—	—
302	i.obs	PD[7]	IO	—
303	o.pin	PD[7]	IO	g5.ctl
304	IO.ctl	G5.CTL	—	—

Table 21-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
305	i.obs	PD[8]	IO	—
306	o.pin	PD[8]	IO	g4.ctl
307	IO.ctl	G4.CTL	—	—
308	i.obs	PD[9]	IO	—
309	o.pin	PD[9]	IO	g3.ctl
310	IO.ctl	G3.CTL	—	—
311	i.obs	PD[10]	IO	—
312	o.pin	PD[10]	IO	g2.ctl
313	IO.ctl	G2.CTL	—	—
314	i.obs	PD[11]	IO	—
315	o.pin	PD[11]	IO	g1.ctl
316	IO.ctl	G1.CTL	—	—
317	i.obs	PD[12]	IO	—
318	o.pin	PD[12]	IO	g81.ctl
319	IO.ctl	G81.CTL	—	—
320	i.obs	PD[13]	IO	—
321	o.pin	PD[13]	IO	g80.ctl
322	IO.ctl	G80.CTL	—	—
323	i.obs	PD[14]	IO	—
324	o.pin	PD[14]	IO	g79.ctl
325	IO.ctl	G79.CTL	—	—
326	i.obs	PD[15]	IO	—
327	o.pin	PD[15]	IO	g78.ctl
328	IO.ctl	G78.CTL	—	—
329	i.obs	PC[4]	IO	—
330	o.pin	PC[4]	IO	g26.ctl
331	IO.ctl	G26.CTL	—	—
332	i.obs	PC[5]	IO	—
333	o.pin	PC[5]	IO	g27.ctl
334	IO.ctl	G27.CTL	—	—
335	i.obs	PB[16]	IO	—
336	o.pin	PB[16]	IO	g40.ctl
337	IO.ctl	G40.CTL	—	—
338	i.obs	PC[6]	IO	—

Table 21-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
339	o.pin	PC[6]	IO	g28.ctl
340	IO.ctl	G28.CTL	—	—
341	i.obs	PB[17]	IO	—
342	o.pin	PB[17]	IO	g47.ctl
343	IO.ctl	G47.CTL	—	—
344	i.obs	PC[7]	IO	—
345	o.pin	PC[7]	IO	g29.ctl
346	IO.ctl	G29.CTL	—	—
347	i.obs	PA[4]	IO	—
348	o.pin	PA[4]	IO	g66.ctl
349	IO.ctl	G66.CTL	—	—
350	i.obs	PB[18]	IO	—
351	o.pin	PB[18]	IO	g48.ctl
352	IO.ctl	G48.CTL	—	—
353	i.obs	PA[5]	IO	—
354	o.pin	PA[5]	IO	g67.ctl
355	IO.ctl	G67.CTL	—	—
356	i.obs	PB[19]	IO	—
357	o.pin	PB[19]	IO	g49.ctl
358	IO.ctl	G49.CTL	—	—
359	i.obs	PA[6]	IO	—
360	o.pin	PA[6]	IO	g68.ctl
361	IO.ctl	G68.CTL	—	—
362	i.obs	PC[8]	IO	—
363	o.pin	PC[8]	IO	g30.ctl
364	IO.ctl	G30.CTL	—	—
365	i.obs	PA[7]	IO	—
366	o.pin	PA[7]	IO	g69.ctl
367	IO.ctl	G69.CTL	—	—
368	i.obs	PC[9]	IO	—
369	o.pin	PC[9]	IO	g31.ctl
370	IO.ctl	G31.CTL	—	—
371	i.obs	PA[8]	IO	—
372	o.pin	PA[8]	IO	g70.ctl

Table 21-1. Boundary Scan Bit Definition (Continued)

BIT	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
373	IO.ctl	G70.CTL	—	—
374	i.obs	PB[22]	IO	—
375	o.pin	PB[22]	IO	g52.ctl
376	IO.ctl	G52.CTL	—	—
377	i.obs	PC[10]	IO	—
378	o.pin	PC[10]	IO	g32.ctl
379	IO.ctl	G32.CTL	—	—
380	i.obs	PA[9]	IO	—
381	o.pin	PA[9]	IO	g71.ctl
382	IO.ctl	G71.CTL	—	—
383	i.obs	PB[23]	IO	—
384	o.pin	PB[23]	IO	g53.ctl
385	IO.ctl	G53.CTL	—	—
386	i.obs	PC[11]	IO	—
387	o.pin	PC[11]	IO	g33.ctl
388	IO.ctl	G33.CTL	—	—
389	i.obs	PB[24]	IO	—
390	o.pin	PB[24]	IO	g54.ctl
391	IO.ctl	G54.CTL	—	—
392	i.obs	PB[25]	IO	—
393	o.pin	PB[25]	IO	g55.ctl
394	IO.ctl	G55.CTL	—	—
395	i.obs	SPARE2	IO	—
396	o.pin	SPARE2	IO	g88.ctl
397	IO.ctl	G88.CTL	—	—

21.3 THE INSTRUCTION REGISTER

The MPC823 JTAG implementation includes the public instructions—**extest**, **sample/preload**, **bypass**, and **clamp**. One additional public instruction—**hi-z**—is capable of disabling all device output drivers. The MPC823 includes a 4-bit instruction register that consists of a shift register with four parallel outputs. Data is transferred from the shift register to the parallel outputs during the update-IR controller state. The four bits are used to decode the five unique instructions listed in Table 21-2.

Table 21-2. Instruction Decoding

CODE				INSTRUCTION
B3	B2	B1	B0	
0	0	0	0	EXTEST
0	0	0	1	SAMPLE/PRELOAD
0	X	1	X	BYPASS
0	1	0	0	HI-Z
	1	0	1	CLAMP and BYPASS

NOTE: B0 (LSB) is shifted first.

The parallel output of the instruction register is reset to all ones in the test-logic-reset controller state. Notice that this preset state is equivalent to the **bypass** instruction. During the capture-IR controller state, the parallel inputs to the instruction shift register are loaded with the **clamp** command code.

21.3.1 The External Test Instruction

The external test (**extest**) instruction selects the 397-bit boundary scan register and asserts an internal reset for the MPC823 system logic to force a known beginning internal state while performing external boundary scan operations. By using the TAP controller, the register is capable of scanning user-defined values into the output buffers, capturing values presented to the input pins, and controlling the output drive of three-stateable output or bidirectional pins. For more details on the function and use of **extest**, refer to the IEEE 1149.1 standard.

21.3.2 The sample/preload Instruction

The **sample/preload** instruction initializes the boundary scan register output cells before **extest** is selected. This initialization ensures that known data will appear on the outputs when entering the **extest** instruction. The **sample/preload** instruction also provides an opportunity to obtain a snapshot of system data and control signals.

Note: Since there is no internal synchronization between the TCK and CLKOUT, you must provide some form of external synchronization between the JTAG operation TCK frequency and system operation CLKOUT frequency to achieve meaningful results.

21.3.3 The bypass Instruction

The **bypass** instruction creates a shift register path from TDI to the bypass register and, finally, to the TDO pins, thus circumventing the 397-bit boundary scan register. This instruction is used to enhance test efficiency when a component other than the MPC823 is the device being tested. It selects the single-bit bypass register as illustrated in Figure 21-7.

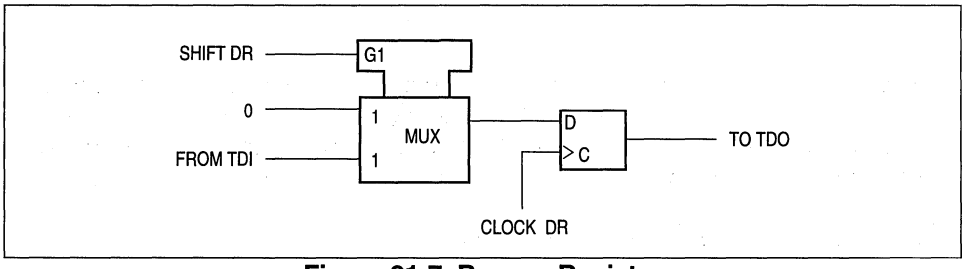


Figure 21-7. Bypass Register

When the bypass register is selected by the current instruction, the shift register stage is set to a logic zero on the rising edge of the TCK pin in the capture-DR controller state. Therefore, the first bit to be shifted out after selecting the bypass register is always a logic zero.

21.3.4 The clamp Instruction

The **clamp** instruction selects the single-bit bypass register as illustrated in Figure 21-7 above, and the state of all signals driven from the system output pins is completely defined by the data previously shifted into the boundary scan register.

21.3.5 The hi-z Instruction

The **hi-z** instruction is provided as a manufacturer's optional public instruction/On to avoid back driving the output pins during circuit board testing. When **hi-z** is invoked all output drivers, including the two-state drivers, are turned off (high impedance). The instructIon selects the bypass register.

21.4 MPC823 RESTRICTIONS

The control afforded by the output enable signals using the boundary scan register and the **extest** instruction requires a compatible circuit board test environment to avoid device-destructive configurations. You must avoid situations in which the MPC823 output drivers are enabled into actively driven networks. The MPC823 features a low-power stop mode. The interaction of the scan chain interface with low-power stop mode is as follows:

1. The TAP controller must be in the test-logic-reset state to either enter or remain in the low-power stop mode. Leaving the TAP controller in the test-logic-reset state negates its ability to achieve low power, but does not otherwise affect device functionality.
2. The TCK input is not disabled in low-power stop mode. To consume minimal power, the TCK input should be externally connected to V_{CC} or ground while in low-power or normal mode (nonscan chain).
3. The TMS, TDI, and $\overline{\text{TRST}}$ pins include on-chip pull-up resistors. In low-power stop mode, the TMS and TDI pins should remain either unconnected or connected to V_{CC} to achieve minimal power consumption. For proper reset of the scan chain test logic, the best approach is to pull active $\overline{\text{TRST}}$ at power-on reset. The easiest way to reset the scan chain logic is to connect $\overline{\text{TRST}}$ to $\overline{\text{HRESET}}$, $\overline{\text{SRESET}}$, or $\overline{\text{PORESET}}$. In power-down mode, you must ensure that the xRESET line is low during power-down so that you can conserve power. If the TAP controller is not used, you should ground the TRST signal. If, for some reason, the $\overline{\text{HRESET}}$ signal that you connected to $\overline{\text{TRST}}$ is high during power-down, the KAPWR supply will propagate through the $\overline{\text{TRST}}$ pin to undesired internal circuits and will increase power consumption.

SECTION 22
ELECTRICAL CHARACTERISTICS

This section contains basic information about power considerations, thermal characteristics, layout practices, and DC timing specifications for the MPC823.

Note: So that we may provide you with the most current information, specific electrical information can be accessed from our website at www.mot.com/mpc823. If you do not have web access, please contact your local Motorola sales office for a printed copy.

22.1 MAXIMUM RATINGS (GND = 0V)

RATING	SYMBOL	VALUE	UNIT
Supply Voltage	VDDH	-0.3 to 4.0	V
	VDD	-0.3 to 4.0	V
	KAPWR	-0.3 to 4.0	V
	VDDSYN	-0.3 to 4.0	V
Input Voltage (JTAG and GPIO)	VIN	-0.3 to 5.8	V
Input Voltage (All other pins)	VIN	-0.3 to 3.3	V
Operating Temperature	T_A	0 to 70° or -40° to 85°	°C
Storage Temperature Range	T_{STG}	-55 to +150	°C

1. Functional operating conditions are given in **Section 22.4 DC Electrical Characteristics (VCC = 3.0 - 3.6 V)**. Absolute maximum ratings are stress ratings only, and functional operation at the maxima is not guaranteed. Stress beyond those listed may affect device reliability or cause permanent damage to the device.
2. **CAUTION**: The JTAG and GPIO input voltages cannot be more than 2.5 V greater than supply voltage, this restriction applies also on "power-on" as well as on normal operation.
3. 5 Volt friendly inputs are inputs that tolerate 5 volts for JTAG and GPIO pins.
4. If you are using Mask Revision Base #F98S (Revision 0), all pins except EXTAL and CLK4IN are 5V tolerant inputs.

This device contains circuitry protecting against damage from high-static voltage or electrical fields. However, it is advised that precautions be taken to avoid application of any voltages higher than the maximum-rated voltages to this high-impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage level (either GND or V_{CC}).

22.2 THERMAL CHARACTERISTICS

CHARACTERISTIC	SYMBOL	VALUE	UNIT
Thermal Resistance for BGA	θ_{Jc}	~30	°C/W

22.3 POWER CONSIDERATIONS

The average chip-junction temperature, T_J, in °C can be obtained from

$$T_J = T_A + (P_D \bullet q_{JA}) \qquad (1)$$

where

T_A = Ambient Temperature, ∞C
q_{JA} = Package Thermal Resistance, Junction to Ambient, ∞C/W
P_D = $P_{INT} + P_{I/O}$
P_{INT} = $I_{DD} \times V_{DD}$, Watts—Chip Internal Power
$P_{I/O}$ = Power Dissipation on Input and Output Pins—User Determined

For most applications $P_{I/O} < 0.3 \bullet P_{INT}$ and can be neglected. If $P_{I/O}$ is neglected, an approximate relationship between P_D and T_J is:

$$P_D = K \prod (T_J + 273\infty C) \qquad (2)$$

Solving equations (1) and (2) for K gives

$$K = P_D \bullet (T_A + 273\infty C) + q_{JA} \bullet P_D^2 \qquad (3)$$

where K is a constant pertaining to the particular part. K can be determined from equation (3) by measuring P_D (at equilibrium) for a known T_A. Using this value of K, the values of P_D and T_J can be obtained by solving equations (1) and (2) iteratively for any value of T_A.

22.3.1 Layout Practices

Each V_{CC} pin on the MPC823 should be provided with a low-impedance path to the board's supply. Each GND pin should be provided with a low-impedance path to ground. The power supply pins drive distinct groups of logic on chip. The V_{CC} power supply should be bypassed to ground using at least four 0.1 μF bypass capacitors located as close as possible to the four sides of the package. The capacitor leads and associated printed circuit traces connecting to chip V_{CC} and GND should be kept to less than half an inch per capacitor lead. A four-layer board is recommended, employing two inner layers as V_{CC} and GND planes.

All output pins on the MPC823 have fast rise and fall times. Printed circuit (PC) trace interconnection length should be minimized in order to minimize undershoot and reflections caused by these fast output switching times. This recommendation particularly applies to the address and data busses. Maximum PC trace lengths of six inches are recommended. Capacitance calculations should consider all device loads as well as parasitic capacitances due to the PC traces. Attention to proper PCB layout and bypassing becomes especially critical in systems with higher capacitive loads because these loads create higher transient currents in the V_{CC} and GND circuits. Pull up all unused inputs or signals that will be inputs during reset. Take special care to minimize the noise levels on the PLL supply pins.

22.4 DC ELECTRICAL CHARACTERISTICS (V_{CC} = 3.0 - 3.6 V)

CHARACTERISTIC	SYMBOL	MIN	MAX	UNIT
Input High Voltage (for JTAG and GPIO)	V_{IH}	2.0	5.5	V
Input High Voltage (all other pins)	V_{IH}	2.0	3.6	V
Input Low Voltage	V_{IL}	GND	0.8	V
EXTAL and EXTCLK Input High Voltage	V_{IHC}	0.7*(V_{CC})	V_{CC}+0.3	V
Input Leakage Current, V_{IN} = 5.5 V	I_{IN}	—	±10	µA
Hi-z (Off State) Leakage Current, V_{IN} = 3.5V	I_{OZ}	—	±10	µA
Signal Low Input Current, V_{IL} = 0.8 V	I_L		±10	µA
Signal High Input Current, V_{IH} = 2.0 V	I_H		±10	µA
Output High Voltage, I_{OH} = –2.0 mA, V_{DDH} = 3.0V Except XTAL, XFC, and Open-Drain Pins	V_{OH}	2.4	—	V
Output Low Voltage IOL = 2.0 mA CLKOUT IOL = 3.2 mA A[6:31], TSIZ0/REG, TSIZ1, D(0:31), DP[0:3]/IRQ[3:6], RD/WR, BURST, RSV/IRQ2, IP_B[0:1]/IWP[0:1]/VFLS[0:1], IP_B2/IOIS16_B/AT2, IP_B3/IWP2/VF2, IP_B4/LWP0/VF0, IP_B5/LWP1/VF1, IP_B6/DSDI/AT0, IP_B7/PTR/AT3, USBRXD/PA15, RXD2/PA13, SMRXD2/L1TXDA/PA9, SMTXD2/L1RXDA/PA8, IRQ4/KR/SPKROUT, TIN1/L1RCLKA/BRGO1/CLK1/PA7, TIN3/TOUT1/CLK2/PA6, TIN2/L1TCLKA/BRGO2/CLK3/PA5, TIN4/TOUT2/CLK4/PA4, LCD_A/SPISEL/PB31, SPICLK/PB30, SPIMOSI/PB29, BRGO3/SPIMISO/PB28, BRGO1/I2CSDA/PB27, BRGO2/I2CSCL/PB26, SMTXD1/PB25, SMRXD1/PB24, SMSYN1/SDACK1/PB23, SMSYN2/SDACK2/PB22, LCD_B/L1ST1/PB19, L1ST2/RTS2/PB18, LCD_C/L1ST3/PB17, L1ST4/L1RQA/PB16, L1ST5/DREQ1/PC15, L1ST6/RTS2/DREQ2/PC14, L1ST7/PC13, L1ST8/L1RQA/PC12, USBRXP/PC11, USBRXN/TGATE1/PC10, CTS2/PC9, TGATE1/CD2/PC8, USBTXP/PC7, USBTXN/PC6, SDACK1/L1TSYNCA/PC5, L1RSYNCA/PC4, LD8/VD7/PD15, LD7/VD6/PD14, LD6/VD5/PD13, LD5/VD4/PD12, LD4/VD3/PD11, LD3/VD2/PD10, LD2/VD1/PD9, LD1/VD0/PD8, FRAME/VSYNC/PD5, LCD_AC/LOE/BLANK/PD6, LD0/FIELD/PD7, LOAD/HSYNC/PD4, SHIFT/CLK/PD3	V_{OL}	—	0.5	V
IOL = 5.3 mA BDIP/GPL_B5, BR, BG, FRZ/IRQ6, CS[0:5], CS6/CE1_B, CS7/CE2_B, WE0/BS_AB0/IORD, WE1/BS_AB1/IOWR, WE2/BS_AB2/PCOE, WE3/BS_AB3/PCWE, GPL_A0/GPL_B0, OE/GPL_A1/GPL_B1, GPL_A[2:3]/GPL_B[2:3]/CS[2:3], UPWAITA/GPL_A4/AS, UPWAITB/GPL_B4, GPL_A5, ALE_B/DSCK/AT1, OP2/MODCK1/STS, OP3/MODCK2/DSDO IOL = 7.0 mA USBOE/PA14, TXD2/PA12 IOL = 8.9 mA TS, TA, TEA, BI, BB, HRESET, SRESET				

NOTE: Input pin voltage specifications are V_{CC} = +4 V or 5.8 V, whichever is less.
AC timings are based on a 50 p*f* load.
If you are using Mask Revision Base #F98S, all pins except EXTAL and CLK4IN are 5V tolerant inputs.

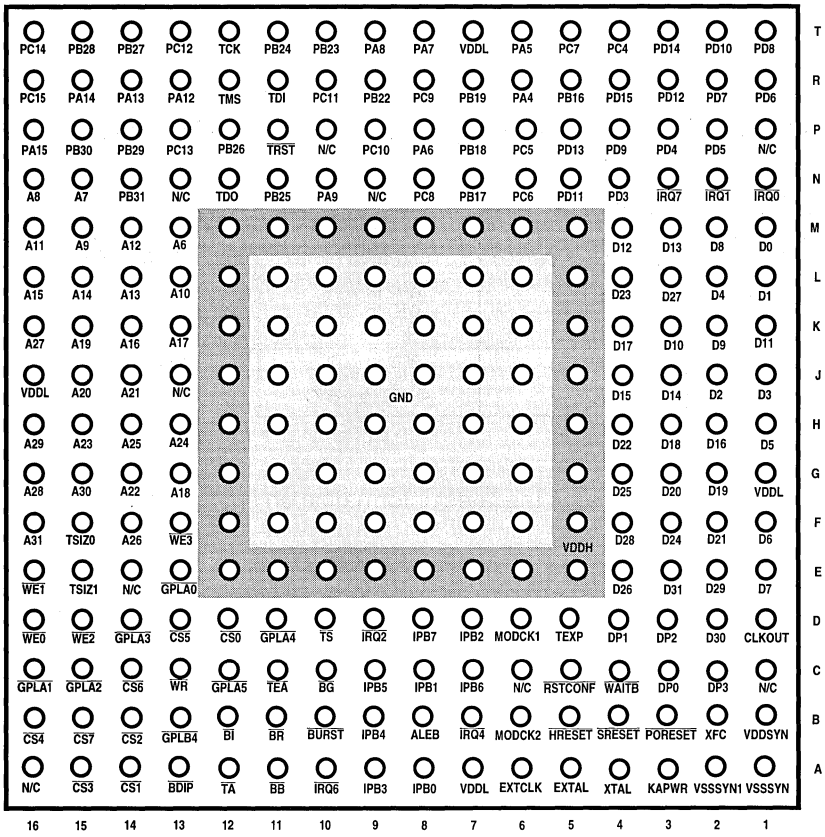
SECTION 23
MECHANICAL DATA AND ORDERING INFORMATION

23.1 ORDERING INFORMATION

PACKAGE TYPE	FREQUENCY (MHz)	TEMPERATURE	ORDER NUMBER
256-Lead Plastic Ball Grid Array with 1.27mm Ball Pitch	25 50 25	0°C to 70°C 0°C to 70°C -45°C to -85°C	XPC823ZT25 XPC823ZT50 XPC823CZT25
256-Lead Plastic Ball Grid Array with 1.00mm Ball Pitch	25 50 25	0°C to 70°C 0°C to 70°C -45°C to -85°C	XPC823ZC25 XPC823ZC50 XPC823CZC25

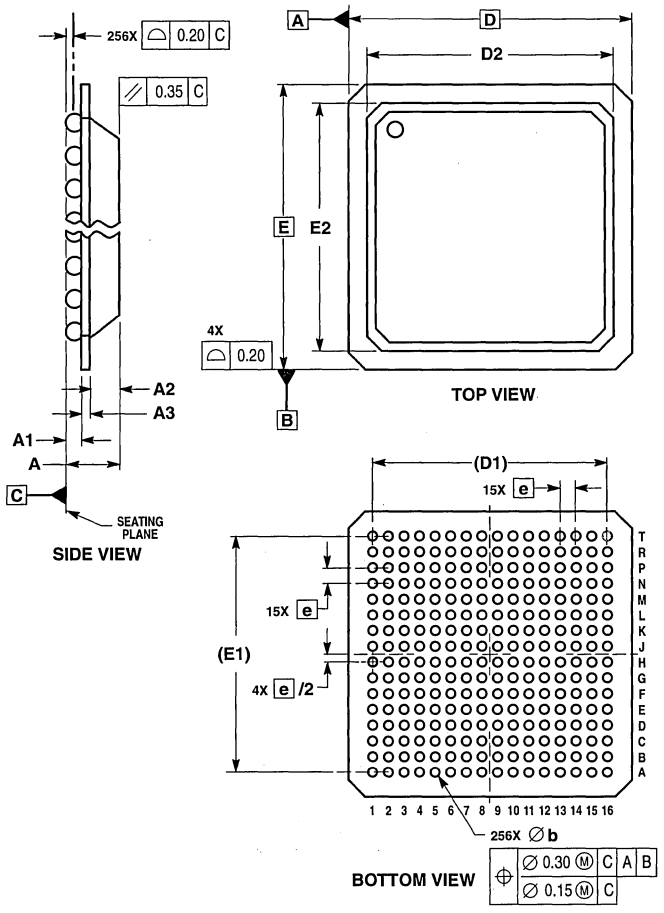
Contact your local Motorola sales office for the latest information on speed grades and part numbers.

23.2 PIN ASSIGNMENTS—PBGA—TOP VIEW



23.3 PACKAGE DIMENSIONS

The following figure is a 23x23mm package, which has 1.27mm spacing between pads. The device designator for the MPC823 in this package is ZT. For more information on the printed circuit board layout of the plastic ball grid array (PBGA) package, including thermal via design and suggested pad layout, please refer to AN-1231/D, *Plastic Ball Grid Array Application Note* available from your local Motorola sales office.

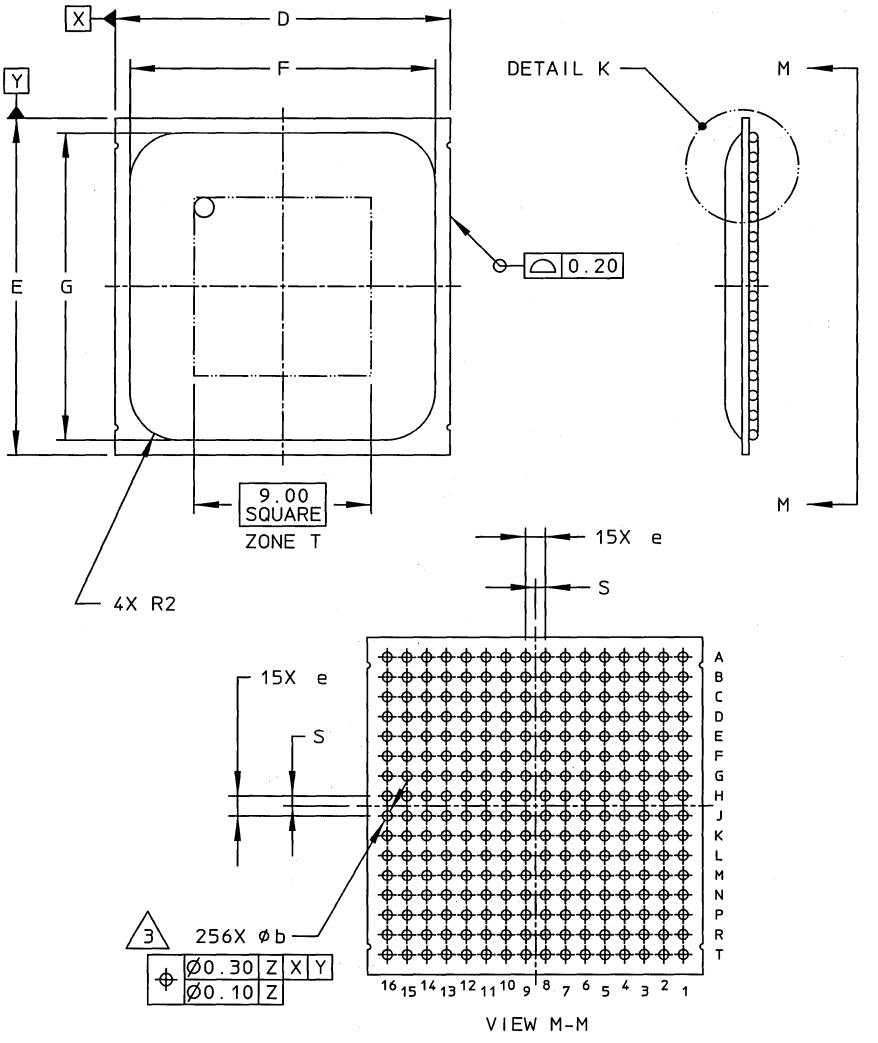


NOTES:
1. DIMENSIONING AND TOLERANCING PER ASME Y14.5M, 1994.
2. DIMENSIONS IN MILLIMETERS.
3. DIMENSION b IS MEASURED AT THE MAXIMUM SOLDER BALL DIAMETER, PARALLEL TO PRIMARY DATUM C.
4. PRIMARY DATUM C AND THE SEATING PLANE ARE DEFINED BY THE SPHERICAL CROWNS OF THE SOLDER BALLS.

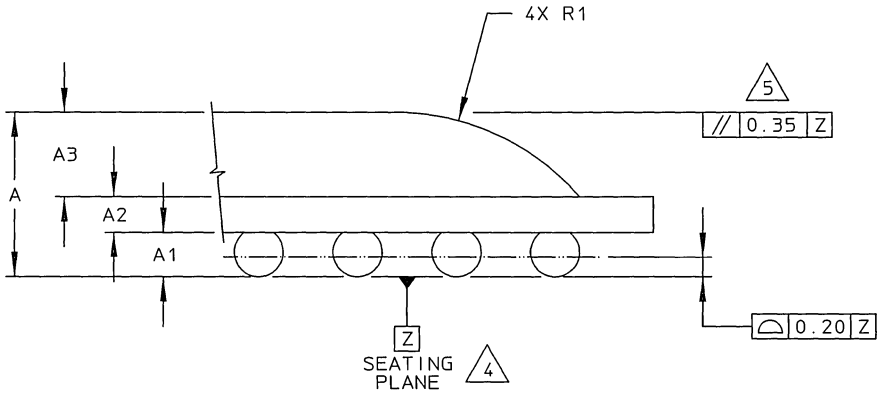
	MILLIMETERS	
DIM	MIN	MAX
A	1.91	2.35
A1	0.50	0.70
A2	1.12	1.22
A3	0.29	0.43
b	0.60	0.90
D	23.00 BSC	
D1	19.05 REF	
D2	19.00	20.00
E	23.00 BSC	
E1	19.05 REF	
E2	19.00	20.00
e	1.27 BSC	

CASE 1130-01
ISSUE A

The following figures are mechanical drawings for the PBGA GTPAC, which is a 17x17mm package that has 1mm spacing between pads. The device designator for the MPC823 in this package is ZC.



CASE NUMBER 1216-02



DETAIL K
ROTATED 90° CLOCKWISE

DIM	MIN	MAX
A	---	1.75
A1	0.27	0.48
A2	0.36 REF	
A3	0.70	1.00
b	0.35	0.65
D	17.00 BSC	
E	17.00 BSC	
e	1.00 BSC	
F	14.00	17.00
G	14.00	17.00
R1	2.50 REF	
R2	0.40	2.50
S	0.50 BSC	

NOTES

1. DIMENSIONS ARE IN MILLIMETERS.
2. INTERPRET DIMENSIONS AND TOLERANCES PER ASME Y14.5M-1994.
3. DIMENSION b IS MEASURED AT THE MAXIMUM SOLDER BALL DIAMETER, PARALLEL TO DATUM PLANE Z
4. DATUM Z (SEATING PLANE) IS DEFINED BY THE SPHERICAL CROWNS OF THE SOLDER BALLS.
5. PARALLELISM REQUIREMENT APPLIES TO ZONE T ONLY. PARALLELISM REQUIREMENT SHALL EXCLUDE ANY EFFECT OF LASER MARK ON TOP SURFACE OF PACKAGE.

SECTION 24
TERMINOLOGY

ACRONYMS, MNEMONICS, AND UNITS OF MEASURE

µs—Microsecond
APG—Access Protection Group
ASID—Address Space ID
BD—Buffer Descriptor
BRG—Baud Rate Generator
CCITT—International Telegraph and Telephone Consultative Committee
C/I—Command/Indication
CPM—Communication Processor Module
CRC—Cyclic Redundancy Check
DISFC—Discarded Frame Counter
EBI—External Bus Interface
FCS—Frame Check Sequence
FD—Function Descriptor
G—Gigabyte
GCI—General Circuit Interface
GPCM—General-Purpose Chip-Select Machine
HDLC—High-Level Data Link Control
I^2C—Inter-Integrated Circuit
IDG—Interdialog Gap
IDL—Interchip Digital Link
IDMA—Independent Direct Memory Access
IDN—Integrated Digital Network
IFG—Interframe Gap
IMMR—Internal Memory Map Register
IrDA—Infra-red
IRLAP—Infra-Red Link Access Protocol
JTAG—Joint Test Action Group
KAPWR—Keep-Alive Power
K—Kilobyte
Kbd—Kilobaud
KBps—Kilobytes Per Second
Kbps—Kilobits Per Second

kHz—Kilohertz
LAP-B—Link Access Protocol-Balanced
LAP-D—Link Access Protocol-D Channel
LRU—Least Recently Used
M—Megabyte
MAC—Multiply Accumulate
Mb—Megabit
Mbps—Megabits Per Second
MF—Multiplication Factor
MHz—Megahertz
MIPS—Millions of Instructions Per Second
MMU—Memory Management Unit
NMI—Nonmaskable Interrupt
NMSI—Nonmaskable Serial Interrupt
NTSC—National Television Standards Committee
OSCM—Main Crystal Oscillator
OSI—Open Systems Interconnection
PAL—Phase Alternation Line
PBGA—Plastic Ball Grid Array
PCI—Peripheral Component Interconnect
PCM—Pulse-Code Modulation
PLL—Phase-Locked Loop
POR—Power-On Reset
PPP—Point-to-Point Protocol
PWM—Pulse-Width Modulation
RMS—Root Mean Squared
RX—Receive or Reception
SCC—Serial Communication Controller
SDMA—Serial Direct Memory Access
SDN—Software Defined Network
SMC—Serial Management Controller
SPI—Serial Peripheral Interface
SS7—Signaling System Number 7
TAP—Test Access Port
TDM—Time-Division Multiplex
TFT—Thin Film Transistor
TLB—Translation Lookaside Buffer
TSA—Time-Slot Assigner
TX—Transmit or Transmission
UART—Universal Asynchronous Receiver/Transmitter
UPM—User-Programmable Machine
USB—Universal Serial Bus

TERMINOLOGY

atomic cycle

If multiple bus transactions by a bus master occur in a sequence where the master retains ownership of the bus during the duration of the sequence, thus preventing other master(s) from transferring in the middle of the sequence, the sequence is considered atomic.

autobaud

The process of determining a serial data rate by timing the width of a single bit.

big-endian

Big-endian ordering assigns the lowest address to the highest order (leftmost) eight bits of the scalar. This called big-endian because the big end of the scalar, considered a binary number, comes first in memory.

breakpoint

An event that forces the machine to branch into a breakpoint exception routine.

burst

A bus transfer that has more than one piece of data associated with it.

burst length

The number of data associated with a burst cycle. For example, a burst length of four has four data pieces (four beats) associated with it.

copyback

Updates to external memory are delayed until forced by the user program or a transfer of bus control to an external master. At the time of forced update or relinquishment of the bus, all changes to the cache are written to external memory. Until that time, cache and external memory are not coherent.

critical-data first

This feature allows the data transferred during the burst cycle to be organized where the word or data needed first is the first one to transfer within the burst-data block. The order of transferring can be sequential and usually wraps back to the word (or data) zero. For example, 1→ 2 → 3 → 0 for a sequence of four data with data 1 as the critical data.

critical-word first

The word that the processor wants first.

datastream

A sequence of information to be processed by the core.

exception

An error, unusual condition, or external signal that can set a status bit. It may or may not cause an interrupt, depending on whether or not the corresponding interrupt is enabled.

execution serialization

Instruction issue is halted until all instructions that are currently in progress complete execution, all internal pipeline stages and instruction buffers have emptied, and all outstanding memory transactions have completed.

execution stream

The combination of instructions and data on which the core operates.

fetch serialization

Instruction fetch is halted until all instructions currently in the processor have completed execution, including the prefetched instructions waiting to be issued. The machine after fetch serialization is said to be completely synchronized.

half-word

A half-word consists of 2 bytes or 16 bits.

internal bus

The bus connecting the core and system interface unit.

interrupt

The act of changing the machine state register and other parts of the machine state in response to an exception.

latency

The interval from the time an instruction begins execution until it produces a result that is available for use by a subsequent instruction.

little-endian

Little-endian byte ordering assigns the lowest address to the lowest-order (rightmost) eight bits of the scalar. The little end of the scalar, considered a binary number, comes first in memory.

master

A device on the bus that requests bus ownership and initiates the bus cycles.

memory controller

A functional logic section of the MPC823. Its primary function is to provide the controls for the external bus memories and I/O devices.

no operation (NOP)

An instruction whose sole function is to increment the program counter, but which affects no changes to any registers or memory.

scoreboard

A register tracking system that ensures that values are not pulled from a register before they are updated by a previous instruction.

sequential instruction

Any instruction that is not a flow control instruction and not ISYNC.

slave

A device that responds to the master's address. A slave receives data on a write cycle and gives data to the master on a read cycle.

snoop

The act of monitoring external bus activity by alternate bus masters. By snooping these external accesses, a core can identify accesses to memory locations that contain dirty data and possibly halt activity to supply correct data.

swap

Four byte lanes, reversing (lane 0 to lane 3, lane 1 to lane 2, lane 2 to lane 1 and lane 3 to lane 0).

tablewalk

An index value is used to identify an entry point in a tree structure that is traversed until a pointer is found. The system walks through a table of pointers to its end.

transaction

A bus transaction consists of an address transfer (address phase) and data transfers (data phase).

time-division multiplex (TDM)

Any serial channel that is divided into channels separated by time.

watchpoint

An event that is reported, but does not change the timing of the machine.

word

A word consists of 4 bytes or 32 bits.

writethrough

Continuous updates, as they occur, of external memory so that cache and memory maintain coherency at all times.

APPENDIX A
SERIAL COMMUNICATION PERFORMANCE

Operating at 25MHz, the MPC823 is designed to support unrestricted operation of the high-level data link control (HDLC) or transparent protocol running on the serial communication controller at 2.048Mbps. The MPC823 can also support one Ethernet channel at 10Mbps and one HDLC or transparent channel at 1Mbps. The physical clocking limit of the serial communication controller is higher than the sustained serial bit rate. This limit is given as a 1:2.25 ratio between the sync clock, which is a clock generated in the clock synthesizer that can be as fast as the 25MHz system clock and the serial clock. For example, with a sync clock of 25MHz, the serial communication controller can be clocked at 11.1MHz. This clocking scheme allows the serial communication controller to handle high-speed bursts of data bits for short periods of time subject to the FIFO sizes.

When the serial communication controller is connected to a time-division multiplexed channel using the time-slot assigner on the MPC823, the serial communication controller's physical clocking limit is a 1:2.5 ratio between the sync clock and serial clock. Therefore, the serial communication controller can be connected to a 10.0MHz time-division multiplexed channel with a 25MHz MPC823. This clocking scheme allows it to handle high-speed bursts of data bits for short periods of time subject to the FIFO sizes. Other devices that offer a higher HDLC performance than the MPC823 are the Motorola MC68605 1984 CCITT X.25 LAPB controller and MC68606 CCITT Q.921 multilink LAPD controller. The MC68605 and MC68606 perform the full data-link layer protocol and support various transparent modes within HDLC-framed operation at speeds of at least 10Mbps. The performance figures listed in Table A-1 are for a 25MHz system clock. Notice that, in general, performance scales linearly with the frequency so that a combination of protocols over the MPC823's performance limitation at 25MHz can occur at 50MHz.

A.1 CHANNEL COMBINATIONS

When operating the multiple HDLC/transparent channel protocol, most of the processing load falls on the communication processor module. Protocols other than the multiple HDLC and Transparent protocol rely on the hardware support built into the serial channels. Combining the serial communication controller and the microcontroller allows you to attain throughput rates of up to 8Mbps in HDLC and Transparent mode.

The multiple channel protocol does not rely on the serial communication controller and operates transparently when you perform serial-to-parallel conversion and vice versa. All bit manipulation is performed in the microcontroller software or hardware, which causes the communication processor module to have a heavier load when operating in multiple channel mode. The heavier load occurs even if all time-slots are concatenated to one logical channel.

The following equation and Table A-1 can be used as a guide for calculating the communication processor module's load. The CPM load estimation is not truly linear, so there is a gray area near the maximum limit (defined as 1). To determine the exact load, you should test your operations on target hardware. The CPM load (L) is approximated by the following equation:

$$L = \sum_{i} \frac{D_i}{P_i} \times \frac{25}{f} < 1$$

where:

Di is the user-targeted data rate for the particular CPM controller or protocol.

Pi is the CPM performance factor for the particular CPM controller or protocol. See Table A-1 for specific values.

f is the actual CPM system frequency in megahertz.

Since the equation and table estimates the feasibility of an application used only by the CPM, it is your responsibility to verify the application against other issues (pin multiplexing, parameter RAM, or microcode availability). The MPC823 Engineer's Toolbox located on the MPC823 website (www.mot.com/mpc823) contains a spreadsheet that performs the calculation of the CPM load.

Table A-1. MPC823 Performance Table

CONTROLLER/PROTOCOL	CPM PERFORMANCE FACTOR
SCC	
Ethernet (Half-Duplex)	22,000 Kbps
Ethernet (Full-Duplex)	11,000 Kbps
HDLC (Full-Duplex)	8,000 Kbps
Transparent (Full-Duplex)	8,000 Kbps
UART (Full-Duplex)	2,400 Kbd
Async HDLC (Full-Duplex)	3,000 Kbps
IrDA High-Speed[1]	16,000 Kbps
IrDA Medium-Speed[2]	16,000 Kbps
IrDA Low-Speed[3]	6,000 Kbps
SPI (Half-Duplex)	
SPI Byte	500 Kbps
SPI Word[4]	3,125 Kbps
SMC	
Transparent (Full-Duplex)	1,500 Kbps
UART (Half-Duplex)	440 Kbd
UART (Full-Duplex)	220 Kbd
IDMA[5]	
DMemory->Peripheral	1,600 KBps
DPeripheral->Memory	2,200 KBps
Memory->Memory	5,700 KBps
Memory->Memory (BURST)	10,400 KBps
SMemory->Peripheral	5,000 KBps
SPeripheral->Memory	5,000 KBps
I^2C (Half-Duplex)	
I^2C	2,080 Kbps
USB	
USB High-Speed	24,000 Kbps
USB Low-Speed	24,000 Kbps

1. Same as Transparent Half-Duplex.
2. Same as HDLC Half-Duplex.
3. Same as Asynchronous HDLC Half-Duplex.
4. The SPI controller is in Word mode if the data character is greater than 8 bits in length.
5. IDMA transfer rates are independent of bus cycle length.

A.2 EXAMPLE #1

If the MPC823 is operating at 25MHz with one Ethernet line in half-duplex at 10Mbps, one 38.4Kbd SMC UART, one 57.6Kbd SMC UART, and one USB at 1.5 Mbps, then the following equation applies.

$$\frac{10}{22} + \frac{38.4}{220} + \frac{57.6}{220} + \frac{1.5}{24} + = 0.953$$

Any calculation such as this, which is close to one, is considered to be in a gray area and should be tested with hardware before it is implemented.

A.3 EXAMPLE #2

If a block of 512K is transferred by IDMA in single address mode from memory to a peripheral, one ASYNC HDLC at 1Mbps, and one SMC UART at 38.4Kbd, then the following equation applies when the MPC823 is running at 25MHz.

$$\frac{512}{5000} + \frac{1000}{3000} + \frac{38.4}{220} + = 0.69$$

Note: With IDMA, this process calculates peak communication processor module usage, not the sustained rate. By nature, IDMA transfers occur in random intervals and are not consistent bit rates when compared to the serial channel operation.

A.4 EXAMPLE #3

For one half-duplex Ethernet channel at 10Mbps and one SMC UART at 115.2Kbd with the MPC823 running at 40MHz, the following equation applies.

$$\left[\frac{10}{22} + \frac{115.2}{220}\right] \times \frac{25}{40} = 0.611$$

APPENDIX B
MPC823 INSTRUCTION SET

This section lists the MPC823 instruction set in alphabetical order by mnemonic. Each entry includes the instruction formats and a quick reference legend that provides information like the level(s) of the PowerPC™ architecture in which the instruction can be found, user- or supervisor-level (an instruction is user-level unless the legend specifies otherwise), and the instruction formats. The format diagrams show all valid combinations of the instruction fields. The architecture specification refers to user- and supervisor-level as problem state and privileged state, respectively.

B.1 INSTRUCTION FORMATS

Instructions are four bytes long and word-aligned, so when instruction addresses are presented to the processor (as in branch instructions) the two low-order bits are ignored. Similarly, whenever the processor develops an instruction address, its two low-order bits are zero. Bits 0–5 always specify the primary opcode and many instructions also have an *e*xtended opcode. The remaining bits of the instruction contains one or more fields for the different instruction formats. Some instruction fields are reserved or must contain a predefined value as shown in the individual instruction layouts. If a reserved field does not have all bits cleared, or if a field that must contain a particular value does not contain that value, the instruction form is invalid.

B.2 SPLIT-FIELD NOTATION

Some instruction fields occupy more than one contiguous sequence of bits or a contiguous sequence of bits used in permuted order. These fields are called split fields. Those split fields that represent the concatenation of the sequences from left to right are shown in lowercase letters. For instance, spr and tbr are shown in the following table.

FIELD	DESCRIPTION
spr (11–20)	This field is used to specify a special-purpose register for the **mtspr** and **mfspr** instructions.
tbr (11–20)	This field is used to specify either the time base lower (TBL) or time base upper (TBU).

Split fields that represent the concatenation of the sequences in some order, which need not be left to right are shown in uppercase letters. For instance, MB, ME, and SH are shown in the following table.

B.3 INSTRUCTION FIELDS

The table below shows the instruction fields used in the various instruction formats.

FIELD	DESCRIPTION
AA (30)	Absolute address bit. 0 The immediate field represents an address relative to the current instruction address (CIA). The effective (logical) address of the branch is either the sum of the LI field sign-extended to 32 bits and the address of the branch instruction or the sum of the BD field sign-extended to 32 bits and the address of the branch instruction. 1 The immediate field represents an absolute address. The effective address (EA) of the branch is the LI field sign-extended to 32 bits or the BD field sign-extended to 32 bits. The LI and LD fields are sign-extended to 32.
BD (16–29)	Immediate field specifying a 14-bit signed two's complement branch displacement that is concatenated on the right with 0b00 and sign-extended to 32 bits.
BI (11–15)	This field is used to specify a bit in the CR to be used as the condition of a branch conditional instruction.
BO (6–10)	This field is used to specify options for the branch conditional instructions.
crbA (11–15)	This field is used to specify a bit in the CR to be used as a source.
crbB (16–20)	This field is used to specify a bit in the CR to be used as a source.
CRM (12–19)	This field mask is used to identify the CR fields that are to be updated by the mtcrf instruction.
d (16–31)	Immediate field specifying a 16-bit signed two's complement integer that is sign-extended to 32 bits.
frC (21–25)	NOT USED BY MPC823.
frD (6–10)	NOT USED BY MPC823.
frS (6–10)	NOT USED BY MPC823.
IMM (16–19)	NOT USED BY MPC823.
LI (6–29)	Immediate field specifying a 24-bit signed two's complement integer that is concatenated on the right with 0b00 and sign-extended to 32 bits.
LK (31)	Link bit. 0 Does not update the link register (LR). 1 Updates the LR. If the instruction is a branch instruction, the address of the instruction following the branch instruction is placed into the LR.
MB (21–25) and ME (26–30)	These fields are used in rotate instructions to specify a 32 bit mask.
NB (16–20)	This field is used to specify the number of bytes to move in an immediate string load or store.
OE (21)	This field is used for extended arithmetic to enable setting OV and SO in the XER.
OPCD (0–5)	Primary opcode field
rA (11–15)	This field is used to specify a GPR to be used as a source or destination.
rB (16–20)	This field is used to specify a GPR to be used as a source.
Rc (31)	Record bit. 0 Does not update the condition register (CR). 1 Updates the CR to reflect the result of the operation. For integer instructions, CR bits 0–2 are set to reflect the result as a signed quantity and CR bit 3 receives a copy of the summary overflow bit, XER[SO]. The result as an unsigned quantity or a bit string can be deduced from the EQ bit. Exceptions are referred to as interrupts in the architecture specification.)
rD (6–10)	This field is used to specify a GPR to be used as a destination.
rS (6–10)	This field is used to specify a GPR to be used as a source.

FIELD	DESCRIPTION
SH (16–20)	This field is used to specify a shift amount.
SIMM (16–31)	This immediate field is used to specify a 16-bit signed integer.
SR (12–15)	This field is used to specify one of the 16 segment registers.
TO (6–10)	This field is used to specify the conditions on which to trap.
UIMM (16–31)	This immediate field is used to specify a 16-bit unsigned integer.
XO (21–30, 22–30, 26–30)	Extended opcode field.

B.4 NOTATIONS AND CONVENTIONS

The operation of some instructions is described by a semiformal language or pseudocode. The table below contains a list of pseudocode notations and conventions used throughout this appendix.

NOTATION/ CONVENTION	DEFINITION
←	Assignment
←iea	Assignment of an instruction effective address.
¬	NOT logical operator
*	Multiplication
÷	Division (yielding quotient)
+	Two's-complement addition
–	Two's-complement subtraction, unary minus
=, ≠	Equals and Not Equals relations
<, ≤, >, ≥	Signed comparison relations
. (period)	Update. When used as a character of an instruction mnemonic, a period (.) means that the instruction updates the condition register field.
c	Carry. When used as a character of an instruction mnemonic, a 'c' indicates a carry out in XER[CA].
e	Extended Precision. When used as the last character of an instruction mnemonic, an 'e' indicates the use of XER[CA] as an operand in the instruction and records a carry out in XER[CA].
o	Overflow. When used as a character of an instruction mnemonic, an 'o' indicates the record of an overflow in XER[OV] and CR0[SO] for integer instructions.
<U, >U	Unsigned comparison relations
?	Unordered comparison relation
&, |	AND, OR logical operators
||	Used to describe the concatenation of two values (that is, 010 || 111 is the same as 010111)
⊕, ≡	Exclusive-OR, Equivalence logical operators (for example, (a ∫ b) = (a ≈ ¬ b))

NOTATION/ CONVENTION	DEFINITION
0b*nnnn*	A number expressed in binary format.
0x*nnnn*	A number expressed in hexadecimal format.
(*n*)x	The replication of x, n times (that is, x concatenated to itself n – 1 times). (n)0 and (n)1 are special cases. A description of the special cases follows: • (n)0 means a field of n bits with each bit equal to 0. Thus (5)0 is equivalent to 0b00000. • (n)1 means a field of n bits with each bit equal to 1. Thus (5)1 is equivalent to 0b11111.
(**r**A|0)	The contents of rA if the rA field has the value 1–31, or the value 0 if the rA field is 0.
(**r**X)	The contents of rX
x[*n*]	n is a bit or field within x, where x is a register
x^n	x is raised to the nth power
ABS(x)	Absolute value of x
CEIL(x)	Least integer ≥ x
Characterization	Reference to the setting of status bits in a standard way that is explained in the text.
CIA	Current instruction address. The 32-bit address of the instruction being described by a sequence of pseudocode. Used by relative branches to set the next instruction address (NIA) and by branch instructions with LK = 1 to set the link register. Does not correspond to any architected register.
Clear	Clear the leftmost or rightmost n bits of a register to 0. This operation is used for rotate and shift instructions.
Clear left and shift left	Clear the leftmost b bits of a register, then shift the register left by n bits. This operation can be used to scale a known non-negative array index by the width of an element. These operations are used for rotate and shift instructions.
Cleared	Bits are set to 0.
Do	Do loop. • Indenting shows range. • "To" and/or "by" clauses specify incrementing an iteration variable. • "While" clauses give termination conditions.
Extract	Select a field of n bits starting at bit position b in the source register, right or left justify this field in the target register, and clear all other bits of the target register to zero. This operation is used for rotate and shift instructions.
EXTS(x)	Result of extending x on the left with sign bits
GPR(x)	General-purpose register x
if...then...else...	Conditional execution, indenting shows range, else is optional.
Insert	Select a field of n bits in the source register, insert this field starting at bit position b of the target register, and leave other bits of the target register unchanged. (No simplified mnemonic is provided for insertion of a field when operating on double words; such an insertion requires more than one instruction.) This operation is used for rotate and shift instructions. (Note that simplified mnemonics are referred to as extended mnemonics in the architecture specification.)
Leave	Leave innermost do loop, or the do loop described in leave statement.
MASK(x, y)	Mask having ones in positions x through y (wrapping if x > y) and zeros elsewhere.
MEM(x, y)	Contents of y bytes of memory starting at address x.

NOTATION/ CONVENTION	DEFINITION
NIA	Next instruction address, which is the 32-bit address of the next instruction to be executed (the branch destination) after a successful branch. In pseudocode, a successful branch is indicated by assigning a value to NIA. For instructions which do not branch, the next instruction address is CIA + 4. Does not correspond to any architected register.
OEA	PowerPC operating environment architecture
Rotate	Rotate the contents of a register right or left n bits without masking. This operation is used for rotate and shift instructions.
Set	Bits are set to 1.
Shift	Shift the contents of a register right or left n bits, clearing vacated bits (logical shift). This operation is used for rotate and shift instructions.
SPR(x)	Special-purpose register x
TRAP	Invoke the system trap handler.
Undefined	An undefined value. The value may vary from one implementation to another, and from one execution to another on the same implementation.
UISA	PowerPC user instruction set architecture
VEA	PowerPC virtual environment architecture

The table below describes instruction field notation conventions used in this appendix.

THE ARCHITECTURE SPECIFICATION	EQUIVALENT
BA, BB, BT	**crb**A, **crb**B, **crb**D (respectively)
D	d
DS	ds
FXM	CRM
RA, RB, RT, RS	**r**A, **r**B, **r**D, **r**S (respectively)
SI	SIMM
U	IMM
UI	UIMM
/, //, ///	0...0 (shaded)

Precedence rules for pseudocode operators are summarized in the following table.

OPERATORS	ASSOCIATIVITY
x[*n*], function evaluation	Left to right
(*n*)x or replication, x(*n*) or exponentiation	Right to left
unary –, ¬	Right to left
*, ÷	Left to right
+, –	Left to right
||	Left to right
=, ≠, <, ≤, >, ≥, <U, >U, ?	Left to right
&, ⊕, ≡	Left to right
|	Left to right
– (range)	None
←, ←iea	None

Operators in the top part of the table above are applied before those in the lower part. Operators at the same level in the table associate from left to right, from right to left, or not at all. For example, "–" (unary minus) associates from left to right, so a – b – c = (a – b) – c. Parentheses are used to override the evaluation order implied by the table above, or to increase clarity and parenthesized expressions are evaluated before serving as operands.

B.5 THE MPC823 INSTRUCTION SET

The remainder of this section lists and describes the MPC823 instruction set. The instructions are listed in alphabetical order by their mnemonic.

Note: The execution unit that executes the instruction may not be the same for all PowerPC processors.

add

Assembler Syntax

add	**r**D,r**A**,r**B** (OE = 0 Rc = 0)
add.	**r**D,r**A**,r**B** (OE = 0 Rc = 1)
addo	**r**D,r**A**,r**B** (OE = 1 Rc = 0)
addo.	**r**D,r**A**,r**B** (OE = 1 Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					OE	266									RC

Operation

rD " (rA) + (rB)

Description

The sum (**r**A) + (**r**B) is placed into **r**D. The **add** instruction is preferred for addition because it sets few status bits.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
  The CR0 field may not reflect the "true" (infinitely precise) result if overflow occurs (see XER below).
- XER:
  Affected: SO, OV (if OE = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XO

addc

Assembler Syntax

addc	**r**D,**r**A,**r**B (OE = 0 Rc = 0)
addc.	**r**D,**r**A,**r**B (OE = 0 Rc = 1)
addco	**r**D,**r**A,**r**B (OE = 1 Rc = 0)
addco.	**r**D,**r**A,**r**B (OE = 1 Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					OE	10									RC

Definition Add Carrying

Operation rD " (rA) + (rB)

Description The sum (**r**A) + (**r**B) is placed into **r**D.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
  If the CR0 field may not reflect the "true" (infinitely precise) result if overflow occurs (see XER below).
- XER:
  Affected: CA
  Affected: SO, OV (if OE = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XO

adde

Assembler Syntax

adde	**r**D,**r**A,**r**B (OE = 0 Rc = 0)
adde.r	D,**r**A,**r**B (OE = 0 Rc = 1)
addeor	D,**r**A,**r**B (OE = 1 Rc = 0)
addeo.	**r**D,**r**A,**r**B (OE = 1 Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					OE	138									RC

Definition Add Extended

Operation rD " (rA) + (rB) + XER[CA]

Description The sum (**r**A) + (**r**B) + XER[CA] is placed into **r**D.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
  The CR0 field may not reflect the "true" (infinitely precise) result if overflow occurs (see XER below).
- XER:
  Affected: CA
  Affected: SO, OV (if OE = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XO

addi

Assembler Syntax addi rD,rA,SIMM)

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

Definition Add Immediate

Operation

if rA = 0 then rD " EXTS(SIMM)
else rD " rA + EXTS(SIMM)

Description

The sum (**r**AI0) + SIMM is placed into **r**D.

The **addi** instruction is preferred for addition because it sets few status bits. Note that **addi** uses the value 0, not the contents of GPR0, if **r**A = 0.

Other registers altered:

- None

Simplified mnemonics:

li	**r**D,value	equivalent to	**addi**	**r**D,**0**,value
la	**r**D,disp(**r**A)	equivalent to	**addi**	**r**D,**r**A,disp
subi	**r**D,**r**A,value	equivalent to	**addi**	**r**D,**r**A,–value

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

addic

Assembler Syntax addic **r**D,**r**A,SIMM)

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

Definition Add Immediate Carrying

Operation rD " (rA) + EXTS(SIMM)

Description The sum (**r**A) + SIMM is placed into **r**D.

Other registers altered:

- XER:
  Affected: CA

Simplified mnemonics:

subic	**r**D,**r**A,value	equivalent to	**addic**	**r**D,**r**A,–value

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

addic.

Assembler Syntax addic. rD,rA,SIMM)

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

Definition Add Immediate Carrying and Record

Operation rD " (rA) + EXTS(SIMM)

Description The sum (**r**A) + SIMM is placed into **r**D.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO
  The CR0 field may not reflect the "true" (infinitely precise) result if overflow occurs (see XER below).
- XER:
  Affected: CA
  Simplified mnemonics:

subic.	**r**D,**r**A,value	equivalent to	**addic.**	**r**D,**r**A,–value

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

addis

Assembler Syntax addis rD,rA,SIMM)

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

Definition Add Immediate Shifted

Operation

if rA = 0 then rD " EXTS(SIMM || (16)0)
else rD " (rA) + EXTS(SIMM || (16)0)

Description

The sum (**r**A|0) + (SIMM || 0x0000) is placed into **r**D.

The **addis** instruction is preferred for addition because it sets few status bits. Note that **addis** uses the value 0, not the contents of GPR0, if **r**A = 0.

Other registers altered:

- None

Simplified mnemonics:

lis	**r**D,value	equivalent to	**addis**	**r**D,**0**,value
subis	**r**D,**r**A,value	equivalent to	**addis**	**r**D,**r**A,–value

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

addme

Assembler Syntax

addme	rD,rA	(OE = 0 Rc = 0)
addme.	rD,rA	(OE = 0 Rc = 1)
addmeo	rD,rA	(OE = 1 Rc = 0)
addmeo.	rD,rA	(OE = 1 Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000					OE	234									RC

Definition

Add to Minus One Extended

Operation

rD " (rA) + XER[CA] – 1

Description

The sum (**r**A) + XER[CA] + 0xFFFF_FFFF_FFFF_FFFF is placed into **r**D.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
  The CR0 field may not reflect the "true" (infinitely precise) result if overflow occurs (see XER below).
- XER:
  Affected: CA
  Affected: SO, OV (if OE = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XO

addze

Assembler Syntax

addze	rD,rA	(OE = 0 Rc = 0)
addze.	rD,rA	(OE = 0 Rc = 1)
addzeo	rD,rA	(OE = 1 Rc = 0)
addzeo.	rD,rA	(OE = 1 Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000					OE	202									RC

Definition

Add to Zero Extended

Operation

rD " (rA) + XER[CA]

Description

The sum (**r**A) + XER[CA] is placed into **r**D.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
  The CR0 field may not reflect the "true" (infinitely precise) result if overflow occurs (see XER below).
- XER:
  Affected: CA
  Affected: SO, OV (if OE = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XO

and

Asembler Syntax

and rA,rS,rB (Rc = 0)
and. rA,rS,rB (Rc = 1)

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

Definition

And

Operation

rA ← (**r**S) & (**r**B)

Description

The contents of **r**S are ANDed with the contents of **r**B and the result is placed into **r**A.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

andc

Assembler Syntax

andc	rA,rS,rB (Rc = 0)
andc.	rA,rS,rB (Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					60										RC

Definition

AND with Complement

Operation

rA ← (**r**S) + ¬ (**r**B)

Description

The contents of **r**S are ANDed with the one's complement of the contents of **r**B and the result is placed into **r**A.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

andi.

Assembler Syntax andi. rA,rS,UIMM

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

Definition AND Immediate

Operation **r**A ← (**r**S) & ((16)0 || UIMM)

Description The contents of **r**S are ANDed with 0x0000 || UIMM and the result is placed into **r**A.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

andis.

Assembler Syntax andis. rA,rS,UIMM

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

Definition AND Immediate Shifted

Operation **r**A ← (**r**S) + (UIMM || (16)0)

Description The contents of **r**S are ANDed with UIMM || 0x0000 and the result is placed into **r**A.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

b

Assembler Syntax

b	target_addr (AA = 0 LK = 0)
ba	target_addr (AA = 1 LK = 0)
bl	target_addr (AA = 0 LK = 1)
bla	target_addr (AA = 1 LK = 1)

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

Definition

Branch

Operation

if AA then NIA $\leftarrow_{iea}$ EXTS(LI || 0b00)
else NIA $\leftarrow_{iea}$ CIA + EXTS(LI || 0b00)
if LK then LR $\leftarrow_{iea}$ CIA + 4

Description

target_addr specifies the branch target address.

If AA = 0, then the branch target address is the sum of LI || 0b00 sign-extended and the address of this instruction. If AA = 1, then the branch target address is the value LI || 0b00 sign-extended. If LK = 1, then the effective address of the instruction following the branch instruction is placed into the link register.

Other registers altered:
Affected: Link Register (LR) (if LK = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			I

bc

Assembler Syntax

bc	BO,BI,target_addr (AA = 0 LK = 0)
bca	BO,BI,target_addr (AA = 1 LK = 0)
bcl	BO,BI,target_addr (AA = 0 LK = 1)
bcla	BO,BI,target_addr (AA = 1 LK = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	16						BO					BI				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	BD														AA	LK

Definition

Branch Conditional

Operation

m ← 32
if ¬ BO[2] then CTR ← CTR – 1
ctr_ok ← BO[2] | (BO[3])
cond_ok ← BO[0] | (CR[BI] ≡ BO[1])
if ctr_ok & cond_ok then
if AA then NIA ←iea EXTS(BD || 0b00)
else NIA ←iea CIA + EXTS(BD || 0b00)
if LK then LR ←iea CIA + 4

Description

The BI field specifies the bit in the condition register (CR) to be used as the condition of the branch. The BO field is encoded as described in the table below.

BO	DESCRIPTION
0000*y*	Decrement the count register (CTR), then branch if the condition is FALSE.
0001*y*	Decrement the CTR, then branch if the condition is FALSE.
001*zy*	Branch if the condition is FALSE.
0100*y*	Decrement the CTR, then branch if the condition is TRUE.
0101*y*	Decrement the CTR, then branch if the condition is TRUE.
011*zy*	Branch if the condition is TRUE.
1*z*00*y*	Decrement the CTR, then branch if the decremented CTR ≠ 0.
1*z*01*y*	Decrement the CTR, then branch if the decremented CTR = 0.
1*z*1*zz*	Branch always.

NOTE: In this table, *z* indicates a bit that is ignored. The *z* bits should be cleared. The *y* bit has a hint about whether a conditional branch is likely to be taken.

target_addr specifies the branch target address.

If AA = 0, the branch target address is the sum of BD || 0b00 sign-extended and the address of this instruction. If AA = 1, the branch target address is the value BD || 0b00 sign-extended. If LK = 1, the effective address of the instruction following the branch instruction is placed into the link register.

Other registers altered:

Affected: Count Register (CTR) (if BO[2] = 0)

Affected: Link Register (LR) (if LK = 1)

Simplified mnemonics:

blt	target	equivalent to	**bc**	**12,0,**target
bne	**cr2,**target	equivalent to	**bc**	**4,10,**target
bdnz	target	equivalent to	**bc**	**16,0,**target

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			B

bcctr

Assembler Syntax

bcctr	BO,BI (LK = 0)
bcctrl	BO,BI (LK = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	19						BO					BI				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000						528									LK

Definition

Branch Conditional to Count Register

Operation

cond_ok ← BO[0] | (CR[BI] ≡ BO[1])
if cond_ok then
NIA $\leftarrow_{iea}$ CTR || 0b00
if LK then LR $\leftarrow_{iea}$ CIA + 4

Description

The BI field specifies the bit in the condition register to be used as the condition of the branch. The BO field is encoded as described in the table below.

BO	DESCRIPTION
0000*y*	Decrement the count register (CTR), then branch if the condition is FALSE.
0001*y*	Decrement the CTR, then branch if the condition is FALSE.
001*zy*	Branch if the condition is FALSE.
0100*y*	Decrement the CTR, then branch if the condition is TRUE.
0101*y*	Decrement the CTR, then branch if the condition is TRUE.
011*zy*	Branch if the condition is TRUE.
1*z*00*y*	Decrement the CTR, then branch if the decremented CTR ≠ 0.
1*z*01*y*	Decrement the CTR, then branch if the decremented CTR = 0.
1*z*1*zz*	Branch always.

NOTE: In this table, *z* indicates a bit that is ignored. The *z* bits should be cleared. The *y* bit has a hint about whether a conditional branch is likely to be taken.

The branch target address is CTR || 0b00.

If LK = 1, the effective address of the instruction following the branch instruction is placed into the link register.

If the "decrement and test CTR" option is specified (BO[2] = 0), the instruction form is invalid.

Other registers altered:

Affected: Link Register (LR) (if LK = 1)

Simplified mnemonics:

bltctr		equivalent to	**bcctr**	**12,0**
bnectr	**cr2**	equivalent to	**bcctr**	**4,10**

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XL

bclr

Assembler Syntax

bclr	BO,BI (LK = 0)
bclrl	BO,BI (LK = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	19						BO					BI				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000						16									LK

Definition

Branch Conditional to Link Register

Operation

```
m ← 32
if ¬ BO[2] then CTR ← CTR – 1
ctr_ok ← BO[2] | ((CTR ≠ 0) ⊕ BO[3])
cond_ok ← BO[0] | (CR[BI] ≡ BO[1])
if ctr_ok & cond_ok then
NIA ←iea LR || 0b00
 if LK then LR ←iea CIA + 4
```

Description

The BI field specifies the bit in the condition register to be used as the condition of the branch. The BO field is encoded as described in the table below.

BO	DESCRIPTION
0000*y*	Decrement the CTR, then branch if the condition is FALSE.
0001*y*	Decrement the CTR, then branch if the condition is FALSE.
001*zy*	Branch if the condition is FALSE.
0100*y*	Decrement the CTR, then branch if the condition is TRUE.
0101*y*	Decrement the CTR, then branch if the condition is TRUE.
011*zy*	Branch if the condition is TRUE.
1*z*00*y*	Decrement the CTR, then branch if the decremented CTR ≠ 0.
1*z*01*y*	Decrement the CTR, then branch if the decremented CTR = 0.
1*z*1*zz*	Branch always.

NOTE: In this table, *z* indicates a bit that is ignored. The *z* bits should be cleared. The *y* bit has a hint about whether a conditional branch is likely to be taken.

The branch target address is LR[0-29] || 0b00.
If LK = 1, then the effective address of the instruction following the branch instruction is placed into the link register.

Other registers altered:

Affected: Count Register (CTR) (if BO[2] = 0)

Affected: Link Register (LR) (if LK = 1)

Simplified mnemonics:

bltlr		equivalent to	**bclr**	**12,0**
bnelr	**cr2**	equivalent to	**bclr**	**4,10**
bdnzlr		equivalent to	**bclr**	**16,0**

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XL

cmp

Assembler Syntax cmp crfD,L,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						CRFD			0	L	A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					0000000000										0

Definition Compare

Operation

a ← EXTS(**r**A)
b ← EXTS(**r**B)
if a < b then c ← 0b100
else if a > b then c ← 0b010
else c ← 0b001
CR[4 * **crf**D–4 * **crf**D + 3] ← c || XER[SO]

Description

The contents of **r**A are compared with the contents of **r**B treating the operands as signed integers. The result of the comparison is placed into CR field **crf**D.

Other registers altered:

- Condition Register (CR field specified by operand crfD): Affected: LT, GT, EQ, SO

Simplified mnemonics:

cmpd	**r**A,**r**B	equivalent to	**cmp**	**0,1,r**A,**r**B
cmpw	**cr3,r**A,**r**B	equivalent to	**cmp**	**3,0,r**A,**r**B

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

cmpi

Assembler Syntax cmpi crfD,L,rA,SIMM

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

Definition Compare Immediate

Operation

a ← (**r**A)
if a < EXTS(SIMM) then c ← 0b100
else if a > EXTS(SIMM) then c ← 0b010
else c ← 0b001
CR[4 * **crf**D–4 * **crf**D + 3] ← c || XER[SO]

Description

The contents of **r**A are compared with the sign-extended value of the SIMM field, treating the operands as signed integers. The result of the comparison is placed into CR field **crf**D.

Other registers altered:

- Condition Register (CR field specified by operand crfD): Affected: LT, GT, EQ, SO

Simplified mnemonics:

cmpdi	**r**A,value	equivalent to	**cmpi**	**0,1,r**A,value
cmpwi	**cr3,r**A,value	equivalent to	**cmpi**	**3,0,r**A,value

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

cmpl

Assembler Syntax cmpl crfD,L,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						CRFD			0	L	A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					32										0

Definition Compare Logical

Operation

a ← **r**A
b ← **r**B
if a <U b then c ← 0b100
else if a >U b then c ← 0b010
else c ← 0b001
CR[4 * **crf**D–4 * **crf**D + 3] ← c || XER[SO]

Description

The contents of **r**A are compared with the contents of **r**B, treating the operands as unsigned integers. The result of the comparison is placed into CR field **crf**D.

Other registers altered:

- Condition Register (CR field specified by operand crfD): Affected: LT, GT, EQ, SO

Simplified mnemonics:

cmpld	**r**A,**r**B	equivalent to	**cmpl**	**0,1,r**A,**r**B
cmplw	**cr3,r**A,**r**B	equivalent to	**cmpl**	**3,0,r**A,**r**B

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

cmpli

Assembler Syntax cmpli crfD,L,rA,UIMM

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

Definition Compare Logical Immediate

Operation

a ← (**r**A)
if a <U ((16)0 || UIMM) then c ← 0b100
else if a >U ((16)0 || UIMM) then c ← 0b010
else c ← 0b001
CR[4 * **crf**D–4 * **crf**D + 3] ← c || XER[SO]

Description

The contents of **r**A are compared with 0x0000|| UIMM, treating the operands as unsigned integers. The result of the comparison is placed into CR field **crf**D.

Other registers altered:

- Condition Register (CR field specified by operand crfD): Affected: LT, GT, EQ, SO

Simplified mnemonics:

cmpldi	**r** A,value	equivalent to	**cmpli**	**0,1,r**A,value
cmplwi	**cr3,r**A,value	equivalent to	**cmpli**	**3,0,r**A,value

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

cntlzw

Assembler Syntax

cntlzw rA,rS (Rc = 0)
cntlzw. rA,rS (Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000						26									RC

Definition

Count Leading Zeros Word

Operation

n ← 0
do while n < 32
if **r**S[n] = 1 then leave
n ← n + 1
rA ← n

Description

A count of the number of consecutive zero bits starting at bit 0 of **r**S is placed into **r**A. This number ranges from 0 to 32, inclusive.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO(if Rc = 1)
  If Rc = 1, then LT is cleared in the CR0 field.

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

crand

Assembler Syntax crand crbD,crbA,crbB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	19						CRBD					CRBA				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	CRBB						257									0

Definition Condition Register AND

Operation CR[**crb**D] ← CR[**crb**A] & CR[**crb**B]

The bit in the condition register specified by **crb**A is ANDed with the bit in the condition register specified by **crb**B. The result is placed into the condition register bit specified by **crb**D.

Other registers altered:

- Condition Register:
  Affected: Bit specified by operand crbD

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XL

crandc

Assembler Syntax crandc **crb**D,**crb**A,**crb**B

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	19						CRBD					CRBA				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	CRBB						129									0

Definition Condition Register AND with Complement

Operation CR[**crb**D] ← CR[**crb**A] & ¬ CR[**crb**B]

Description The bit in the condition register specified by **crb**A is ANDed with the complement of the bit in the condition register specified by **crb**B and the result is placed into the condition register bit specified by **crb**D.

Other registers altered:

- Condition Register:
  Affected: Bit specified by operand crbD

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XL

creqv

Assembler Syntax creqv crbD,crbA,crbB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	19						CRBD					CRBA				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	CRBB						289									0

Definition Condition Register Equivalent

Operation CR[**crb**D] ← CR[**crb**A] ≡ CR[**crb**B]

The bit in the condition register specified by **crb**A is XORed with the bit in the condition register specified by **crb**B and the complemented result is placed into the condition register bit specified by **crb**D.

Other registers altered:

- Condition Register:
  Affected: Bit specified by operand crbD

Simplified mnemonics:

crset	**crb**D	equivalent to	**creqv**	**crb**D,**crb**D, **crb**D

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XL

crnand

Assembler Syntax crnand crbD,crbA,crbB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	19						CRBD					CRBA				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	CRBB						225									0

Definition Condition Register NAND

Operation CR[**crb**D] ← ¬ (CR[**crb**A] & CR[**crb**B])

The bit in the condition register specified by **crb**A is ANDed with the bit in the condition register specified by **crb**B and the complemented result is placed into the condition register bit specified by **crb**D.

Other registers altered:

- Condition Register:
  Affected: Bit specified by operand crbD

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XL

crnor

Assembler Syntax crnor crbD,crbA,crbB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	19						CRBD					CRBA				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	CRBB						33									0

Definition Condition Register NOR

Operation CR[**crb**D] ← ¬ (CR[**crb**A] | CR[**crb**B])

The bit in the condition register specified by **crb**A is ORed with the bit in the condition register specified by **crb**B and the complemented result is placed into the condition register bit specified by **crb**D.

Other registers altered:

- Condition Register:
  Affected: Bit specified by operand crbD

Simplified mnemonics:

crnot	**crb**D,**crb**A	equivalent to	**crnor**	**crb**D,**crb**A, **crb**A

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XL

cror

Assembler Syntax cror crbD,crbA,crbB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	19						CRBD					CRBA				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	CRBB						449									0

Definition Condition Register OR

Operation CR[**crb**D] ← CR[**crb**A] | CR[**crb**B]

Description The bit in the condition register specified by **crb**A is ORed with the bit in the condition register specified by **crb**B. The result is placed into the condition register bit specified by **crb**D.

Other registers altered:

- Condition Register:
  Affected: Bit specified by operand crbD

Simplified mnemonics:

crmove	**crb**D,**crb**A	equivalent to	**cror**	**crb**D,**crb**A, **crb**A

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XL

crorc

Assembler Syntax crorc crbD,crbA,crbB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	19						CRBD					CRBA				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	CRBB						417									0

Definition Condition Register OR with Complement

Operation CR[**crb**D] ← CR[**crb**A] | ¬ CR[**crb**B]

Description The bit in the condition register specified by **crb**A is ORed with the complement of the condition register bit specified by **crb**B and the result is placed into the condition register bit specified by **crb**D.

Other registers altered:

- Condition Register:
  Affected: Bit specified by operand crbD

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XL

crxor

Assembler Syntax crxor crbD,crbA,crbB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	19						CRBD					CRBA				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	CRBB						193									0

Definition Condition Register XOR

Operation CR[**crb**D] ← CR[**crb**A] ⊕ CR[**crb**B]

Description The bit in the condition register specified by **crb**A is XORed with the bit in the condition register specified by **crb**B and the result is placed into the condition register specified by **crb**D.

Other registers altered:

- Condition Register:
  Affected: Bit specified by crbD

Simplified mnemonics:

crclr	**crb**D	equivalent to	**crxor**	**crb**D,**crb**D, **crb**D

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XL

dcbf

AssemblerSyntax dcbf rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						00000					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						86									0

Definition Data Cache Block Flush

Operation EA is the sum (**r**AI0) + (**r**B).

Description The **dcbf** instruction invalidates the *block* in the data *cache* addressed by EA, copying the block to memory first, if there is any dirty data in it. If the processor is a multiprocessor implementation and the block is marked coherency-required, the processor will, if necessary, send an address-only broadcast to other processors. The broadcast of the **dcbf** instruction causes another processor to copy the block to memory, if it has dirty data, and then invalidate the block from the cache.

The action taken depends on the memory mode associated with the block containing the byte addressed by EA and on the state of that block. The list below describes the action taken for the various states of the *memory coherency* attribute (M bit).

- Coherency required
  - ❑ Unmodified block—Invalidates copies of the block in the data caches of all processors.
  - ❑ Modified block—Copies the block to memory. Invalidates copies of the block in the data caches of all processors.
  - ❑ Absent block—If modified copies of the block are in the data caches of other processors, causes them to be copied to memory and invalidated in those data caches. If unmodified copies are in the data caches of other processors, causes those copies to be invalidated in those data caches.
- Coherency not required
  - ❑ Unmodified block—Invalidates the block in the processor's data cache.

- Modified block—Copies the block to memory. Invalidates the block in the processor's data cache.
- Absent block (target block not in cache)—No action is taken.

The function of this instruction is independent of the write-through, *write-back* and *caching-inhibited*/allowed modes of the block containing the byte addressed by EA.

This instruction may be treated as a load from the addressed byte with respect to address translation and memory protection. It may also be treated as a load for referenced and changed bit recording except that referenced and changed bit recording may not occur.

Other registers altered:

- None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

dcbi

Asembler Syntax dcbi rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						00000					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						470									0

Definition Data Cache Block Invalidate

Operation EA is the sum (**r**A|0) + (**r**B).

Description The action taken is dependent on the memory mode associated with the block containing the byte addressed by EA and on the state of that block. The list below describes the action taken if the block containing the byte addressed by EA is or is not in the cache.

- Coherency required
  - ❑ Unmodified block—Invalidates copies of the block in the data caches of all processors.
  - ❑ Modified block—Invalidates copies of the block in the data caches of all processors. (Discards the modified contents.)
  - ❑ Absent block—If copies of the block are in the data caches of any other processor, causes the copies to be invalidated in those data caches. (Discards any modified contents.)
- Coherency not required
  - ❑ Unmodified block—Invalidates the block in the processor's data cache.
  - ❑ Modified block—Invalidates the block in the processor's data cache. (Discards the modified contents.)
  - ❑ Absent block (target block not in cache)—No action is taken.

When data address translation is enabled, MSR[DR] = 1, and the *virtual address* has no translation, a DSI *exception* occurs. The function of this instruction is independent of the write-through and caching-inhibited/allowed modes of the block containing the byte addressed by EA. This instruction operates as a store to the addressed byte with respect to address translation and protection. The referenced and changed bits are modified appropriately. This is a supervisor-level instruction.

Other registers altered:

- None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

dcbst

Assembler Syntax dcbst rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						00000					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						54									0

Definition Data Cache Block Store

Operation EA is the sum (**r**AI0) + (**r**B).

Description The dcbst instruction executes as follows:

- If the block containing the byte addressed by EA is in coherency-required mode, and a block containing the byte addressed by EA is in the data cache of any processor and has been modified, the writing of it to main memory is initiated.
- If the block containing the byte addressed by EA is in coherency-not-required mode, and a block containing the byte addressed by EA is in the data cache of this processor and has been modified, the writing of it to main memory is initiated.

The function of this instruction is independent of the write-through and caching-inhibited/allowed modes of the block containing the byte addressed by EA. The processor treats this instruction as a load from the addressed byte with respect to address translation and memory protection. It may also be treated as a load for referenced and changed bit recording except that referenced and changed bit recording may not occur.

Other registers altered:

- None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
VEA			X

dcbt

Assembler Syntax dcbt rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						00000					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						278									0

Definition Data Cache Block Touch

Operation EA is the sum (**r**A|0) + (**r**B).

Description This instruction is a hint that performance will probably be improved if the block containing the byte addressed by EA is *fetched* into the data cache, because the program will probably soon load from the addressed byte. The hint is ignored if the block is caching-inhibited. Executing **dcbt** does not cause the system alignment error handler to be invoked.

This instruction may be treated as a load from the addressed byte with respect to address translation, memory protection, and reference and change recording, except that no exception occurs in the case of a translation fault or protection violation.

The program uses the **dcbt** instruction to request a *cache block* fetch before it is actually needed by the program. The program can later execute load instructions to put data into registers. However, the processor is not obliged to load the addressed block into the data cache.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
VEA			X

dcbtst

Assembler Syntax dcbtst rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						00000					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						246									0

Definition Data Cache Block Touch for Store

Operation EA is the sum (**r**AI0) + (**r**B).

Description This instruction is a hint that performance will be improved if the block containing the byte addressed by EA is fetched into the data cache, because the program will probably soon store into the addressed byte. The hint is ignored if the block is caching-inhibited. Executing **dcbtst** does not cause the system alignment error handler to be invoked.

This instruction operates as a load from the addressed byte with respect to address translation and protection, except that no exception occurs in the case of a translation fault or protection violation. Also, if the referenced and changed bits are recorded, they are recorded as if the access was a load.

The program uses **dcbtst** to request a cache block fetch to guarantee that a subsequent store will be to a cached location. The program can later execute store instructions to put data into memory. However, the processor is not obliged to load the addressed cache block into the data cache.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
VEA			X

dcbz

Assembler Syntax dcbz rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						00000					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						1014									0

Definition Data Cache Block Set to Zero

Operation EA is the sum (**r**AI0) + (**r**B).

Description The dcbz instruction executes as follows:

- If the cache block containing the byte addressed by EA is in the data cache, all bytes are cleared.
- If the cache block containing the byte addressed by EA is not in the data cache and the corresponding page is caching-allowed, the cache block is allocated in the data cache (without fetching the block from main memory), and all bytes are cleared.
- If the page containing the byte addressed by EA is in caching-inhibited or write-through mode, either all bytes of main memory that correspond to the addressed cache block are cleared or the alignment exception handler is invoked. The exception handler clears all bytes in main memory that corresponds to the addressed cache block.
- If the cache block containing the byte addressed by EA is in coherency-required mode, and the cache block exists in the data cache(s) of any other processor(s), it is kept coherent in those caches.

This instruction is treated as a store to the addressed byte with respect to address translation, memory protection, referenced and changed recording and the ordering enforced by **eieio** or by the combination of caching-inhibited and guarded attributes for a page.

Other registers altered:

- None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
VEA			X

divw

Assembler Syntax

divw	rD,rA,rB (OE = 0 Rc = 0)
divw.	rD,rA,rB (OE = 0 Rc = 1)
divwo	rD,rA,rB (OE = 1 Rc = 0)
divwo.	rD,rA,rB (OE = 1 Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					OE	491									RC

Definition

Divide Word

Operation

dividend ← (**r**A)
divisor ← (**r**B)
rD ← dividend ÷ divisor

Description

The dividend is the contents of **r**A. The divisor is the contents of **r**B. The 32-bit quotient is formed and placed in **r**D. The remainder is not supplied as a result.

Both the operands and the quotient are interpreted as signed integers. The quotient is the unique signed integer that satisfies the equation—dividend = (quotient * divisor) + r where $0 \le r <$ |divisor| (if the dividend is non-negative), and $-|\text{divisor}| < r \le 0$ (if the dividend is negative).

If an attempt is made to perform any of the divisions—0x8000_0000 ÷ –1or <anything> ÷ 0—then the contents of **r**D are undefined, as are the contents of the LT, GT, and EQ bits of the CR0 field (if Rc = 1). In this case, if OE = 1 then OV is set.

The 32-bit signed remainder of dividing the contents of rA by the contents of rB can be computed as follows, except in the case that the contents of rA = –231 and the contents of rB = –1.

divw	**r**D,**r**A,**r**B	# **r**D = quotient
mullw	**r**D,**r**D,**r**B	# **r**D = quotient * divisor
subf	**r**D,**r**D,**r**A	# **r**D = remainder

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
- XER:
  Affected: SO, OV (if OE = 1)
  The setting of the affected bits in the XER is mode-independent, and reflects overflow of the 32-bit result.

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XO

divwu

Assembler Syntax

divwu	rD,rA,rB	(OE = 0 Rc = 0)
divwu.	rD,rA,rB	(OE = 0 Rc = 1)
divwuo	rD,rA,rB	(OE = 1 Rc = 0)
divwuo.	rD,rA,rB	(OE = 1 Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					OE	459									RC

Definition

Divide Word Unsigned

Operation

dividend ← (**r**A)
divisor ← (**r**B)
rD ← dividend ÷ divisor

Description

The dividend is the contents of **r**A. The divisor is the contents of **r**B. A 32-bit quotient is formed. The 32-bit quotient is placed into **r**D. The remainder is not supplied as a result.

Both operands and the quotient are interpreted as unsigned integers, except that if Rc = 1 the first three bits of CR0 field are set by signed comparison of the result to zero. The quotient is the unique unsigned integer that satisfies the equation—dividend = (quotient * divisor) + r (where 0 ≤ r < divisor). If an attempt is made to perform the division—<anything> ÷ 0—then the contents of **r**D are undefined as are the contents of the LT, GT, and EQ bits of the CR0 field (if Rc = 1). In this case, if OE = 1 then OV is set.

The 32-bit unsigned remainder of dividing the contents of rA by the contents of rB can be computed as follows:

```
divwu    rD,rA,rB    # rD = quotient
mullw    rD,rD,rB    # rD = quotient * divisor
subf     rD,rD,rA    # rD = remainder
```

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO(if Rc = 1)
- XER:
  Affected: SO, OV(if OE = 1)
  The setting of the affected bits in the XER is mode-independent, and reflects overflow of the 32-bit result.

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XO

eciwx

Assembler Syntax eciwx rD,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						310									0

Definition External Control In Word Indexed

Operation

if **r**A = 0 then b ← 0
else b← (**r**A)
EA ← b + (**r**B)
paddr ← address translation of EA
send load word request for paddr to device identified by EAR[RID]
rD ← word from device

Description

The **eciwx** instruction allows the system designer to map special devices in an alternative way. The MMU translation of the EA is not used to select the special device, as it is used in most instructions such as loads and stores. Rather, it is used as an address operand that is passed to the device over the address bus. Four other pins (the burst and size pins on the 60x bus) are used to select the device; these four pins output the 4-bit resource ID (RID) field that is located in the EAR register. The **eciwx** instruction also loads a word from the data bus that is output by the special device.

The **eciwx** instruction and the EAR register can be very efficient when mapping special devices such as graphics devices that use addresses as pointers.

EA is the sum (**r**A|0) + (**r**B).

A load word request for the physical address (referred to as real address in the architecture specification) corresponding to EA is sent to the device identified by EAR[RID], bypassing the cache. The word returned by the device is placed in **r**D. EAR[E] must be 1. If it is not, a DSI exception is generated.

EA must be a multiple of four. If it is not, one of the following occurs:

- A system alignment exception is generated.
- A DSI exception is generated (possible only if EAR[E] = 0).
- The results are boundedly undefined.

The **eciwx** instruction is supported for EAs that reference memory *segments* in which SR[T] = 1 and for EAs mapped by the DBAT registers. If the EA references a *direct-store* segment (SR[T] = 1), either a DSI exception occurs or the results are boundedly undefined. However, note that the direct-store facility is being phased out of the architecture and will not likely be supported in future devices. Thus, software should not depend on its effects.

If this instruction is executed when MSR[DR] = 0 (real addressing mode), the results are boundedly undefined. This instruction is treated as a load from the addressed byte with respect to address translation, memory protection, referenced and changed bit recording, and the ordering performed by **eieio**. This instruction is optional in the PowerPC architecture.

Other registers altered:

- None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
VEA		√	X

ecowx

Assembler Syntax ecowx rS,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						438									0

Definition External Control Out Word Indexed

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + (**r**B)
paddr ← address translation of EA
send store word request for paddr to device identified by EAR[RID]
send **r**S to device

Description

The **ecowx** instruction and the EAR register can be very efficient when mapping special devices such as graphics devices that use addresses as pointers.

EA is the sum (**r**AI0) + (**r**B). A store word request for the physical address corresponding to EA and the contents of **r**S are sent to the device identified by EAR[RID], bypassing the cache. EAR[E] must be 1, if it is not, a DSI exception is generated. EA must be a multiple of four. If it is not, one of the following occurs:

- A system alignment exception is generated.
- A DSI exception is generated (possible only if EAR[E] = 0).
- The results are boundedly undefined.

The **ecowx** instruction is supported for effective addresses that reference memory segments in which SR[T] = 0, and for EAs mapped by the DBAT registers. If the EA references a direct-store segment (SR[T] = 1), either a DSI exception occurs or the results are boundedly undefined. However, note that the direct-store facility is being phased out of the architecture and will not likely be supported in future devices. Thus, software should not depend on its effects.

If this instruction is executed when MSR[DR] = 0 (real addressing mode), the results are boundedly undefined. This instruction is treated as a store from the addressed byte with respect to address translation, memory protection, nd referenced and changed bit recording, and the ordering performed by **eieio**. This instruction is optional in the PowerPC architecture.

Other registers altered:

- None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
VEA		√	X

eieio

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						00000					00000				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000						854									0

Definition

Enforce In-Order Execution of I/O

Description

The **eieio** instruction provides an ordering function for the effects of load and store instructions executed by a processor. These loads and stores are divided into two sets, which are ordered separately. The memory accesses caused by a **dcbz** instruction are ordered like a store. The two sets are as follows:

- Loads and stores to memory that are caching-inhibited, guarded, and stores to memory that is write-through required. The **eieio** instruction controls the order in which the accesses are performed in main memory. It ensures that all applicable memory accesses caused by instructions preceding the **eieio** instruction have completed with respect to main memory before any applicable memory accesses caused by instructions following the **eieio** instruction access main memory. It acts as a barrier that flows through the memory queues to main memory, preventing the reordering of memory accesses across the barrier. No ordering is performed for **dcbz** if the instruction causes the system alignment error handler to be invoked. All accesses in this set are ordered as a single set—that is, there is not one order for loads and stores to caching-inhibited and guarded memory and another order for stores to write-through required memory.
- Stores to memory that have all of the following attributes—caching-allowed, write-through not required, and memory-coherency required. The **eieio** instruction controls the order in which the accesses are performed with respect to coherent memory. It ensures that all applicable stores caused by instructions preceding the eieio instruction have completed with respect to coherent memory before any applicable stores caused by instructions following the eieio instruction complete with respect to coherent memory. With the exception of **dcbz**, **eieio** does not affect the order of cache operations (whether caused explicitly by execution of

a cache management instruction, or implicitly by the *cache coherency* mechanism). The **eieio** instruction does not affect the order of accesses in one set with respect to accesses in the other set.

The **eieio** instruction may complete before memory accesses caused by instructions preceding the **eieio** instruction have been performed with respect to main memory or coherent memory as appropriate.

The **eieio** instruction is intended for use in managing shared data structures, in accessing memory-mapped I/O, and in preventing load/store combining operations in main memory. For the first use, the shared data structure and the lock that protects it must be altered only by stores that are in the same set (1 or 2; see previous discussion). For the second use, **eieio** can be thought of as placing a barrier into the stream of memory accesses issued by a processor, such that any given memory access appears to be on the same side of the barrier to both the processor and the I/O device. Because the processor performs store operations in order to memory that is designated as both caching-inhibited and guarded, the **eieio** instruction is needed for such memory only when loads must be ordered with respect to stores or with respect to other loads.

Note that the **eieio** instruction does not connect hardware considerations to it, such as multiprocessor implementations that send an **eieio** address-only broadcast (useful in some designs). For example, if a design has an external buffer that reorders loads and stores for better bus efficiency, the **eieio** broadcast signals to that buffer that previous loads/stores (marked caching-inhibited, guarded, or write-through required) must complete before any following loads/stores (marked caching-inhibited, guarded, or write-through required).

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
VEA			X

eqv

Assembler Syntax

eqv	rA,rS,rB (Rc = 0)
eqv.	rA,rS,rB (Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					284										RC

Definition Equivalent

Operation **r**A ← (**r**S) ≡ (**r**B)

Description The contents of **r**S are XORed with the contents of **r**B and the complemented result is placed into **r**A.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
VEA			X

extsb

Assembler Syntax

extsb	rA,rS (Rc = 0)
extsb.	rA,rS (Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000					954										RC

Definition

Extend Sign Byte

Operation

S ← **r**S[24]
rA[24-31] ← **r**S[24-31]
rA[0–23] ← (24)S

Description

The contents of **r**S[24-31] are placed into **r**A[24-31]. Bit 24 of **r**S is placed into **r**A[0-23].

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

extsh

Assembler Syntax

	extsh	rA,rS (Rc = 0)
	extsh.	rA,rS (Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000						922									RC

Definition

Extend Sign Half Word

Operation

S ← **r**S[16]
rA[16-31] ← **r**S[16-31]
rA[0-15] ← (16)S

Description

The contents of **r**S[16-31] are placed into **r**A[16-31]. Bit 16 of **r**S is placed into **r**A[0–15].

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

icbi

Assembler Syntax icbi rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						00000					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						982									0

Definition Instruction Cache Block Invalidate

Description EA is the sum (**r**A|0) + (**r**B).

If the block containing the byte addressed by EA is in coherency-required mode, and a block containing the byte addressed by EA is in the instruction cache of any processor, the block is made invalid in all such instruction caches, so that subsequent references cause the block to be refetched.

If the block containing the byte addressed by EA is in coherency-not-required mode, and a block containing the byte addressed by EA is in the instruction cache of this processor, the block is made invalid in that instruction cache, so that subsequent references cause the block to be refetched. The function of this instruction is independent of the write-through, write-back, and caching-inhibited/allowed modes of the block containing the byte addressed by EA.

This instruction is treated as a load from the addressed byte with respect to address translation and memory protection. It may also be treated as a load for referenced and changed bit recording except that referenced and changed bit recording may not occur. Implementations with a combined data and instruction cache treat the **icbi** instruction as a no-op, except that they may invalidate the target block in the instruction caches of other processors if the block is in coherency-required mode.

The **icbi** instruction invalidates the block at EA (rA|0 + rB). If the processor is a multiprocessor implementation and the block is marked coherency-required, the processor will send an address only broadcast to other processors causing those processors to invalidate the block from their instruction caches.

For faster processing, many implementations will not compare the entire EA (rAI0 + rB) with the tag in the instruction cache. Instead, they will use the bits in the EA to locate the set that the block is in, and invalidate all blocks in that set.

Other registers altered:

- None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
VEA			X

isync

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	19						00000					00000				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000						150									0

Definition

Instruction Synchronize

Description

The **isync** instruction provides an ordering function for the effects of all instructions executed by a processor. Executing an **isync** instruction ensures that all instructions preceding the the **isync** instruction have completed before the **isync** instruction completes, except that memory accesses caused by those instructions need not have been performed with respect to other processors and mechanisms. It also ensures that no subsequent instructions are initiated by the processor until after the **isync** instruction completes. Finally, it causes the processor to discard any prefetched instructions, with the effect that subsequent instructions will be fetched and executed in the context established by the instructions preceding the isync instruction. The **isync** instruction has no effect on the other processors or on their caches. This instruction is context synchronizing.

Context synchronization is necessary after certain code sequences that perform complex operations within the processor. These code sequences are usually operating system tasks that involve memory management. For example, if an instruction "A" changes the memory translation rules in the *memory management unit (MMU)*, the **isync** instruction should be executed so that the instructions following instruction "A" will be discarded from the pipeline and refetched according to the new translation rules. This instruction is context synchronizing.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
VEA			XL

lbz

Assembler Syntax lbz rD,d(rA)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	34						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	d															

Definition Load Byte and Zero

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + EXTS(d)
rD ← (24)0 || MEM(EA, 1)

Description EA is the sum (**r**A|0) + d. The byte in memory addressed by EA is loaded into the low-order eight bits of **r**D. The remaining bits in **r**D are cleared.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

lbzu

Assembler Syntax lbzu rD,d(rA)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	35						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	d															

Definition Load Byte and Zero with Update

Operation

EA ← (**r**A) + EXTS(d)
rD ← (24)0 || MEM(EA, 1)
rA ← EA

Description

EA is the sum (**r**A) + d. The byte in memory addressed by EA is loaded into the low-order eight bits of **r**D. The remaining bits in **r**D are cleared. EA is placed into **r**A. If rA = 0, or rA = rD, the instruction form is invalid.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

lbzux

Assembler Syntax lbzux rD,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						119									0

Definition Load Byte and Zero with Update Indexed

Operation

EA ← (**r**A) + (**r**B)
rD ← (24)0 || MEM(EA, 1)
rA ← EA

Description EA is the sum (**r**A) + (**r**B). The byte in memory addressed by EA is loaded into the low-order eight bits of **r**D. The remaining bits in **r**D are cleared. EA is placed into **r**A. If rA = 0 or rA = rD, the instruction form is invalid.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

lbzx

Assembler Syntax lbzx rD,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						87									0

Definition Load Byte and Zero Indexed

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + (**r**B)
rD ← (24)0 || MEM(EA, 1)

Description EA is the sum (**r**A|0) + (**r**B). The byte in memory addressed by EA is loaded into the low-order eight bits of **r**D. The remaining bits in **r**D are cleared.

Other registers altered:

- None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

lha

Assembler Syntax lha rD,d(rA)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	42						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	d															

Definition Load Half Word Algebraic

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + EXTS(d)
rD ← EXTS(MEM(EA, 2))

Description EA is the sum (**r**A|0) + d. The half word in memory addressed by EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are filled with a copy of the most-significant bit of the loaded half word.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

lhau

Assembler Syntax lhau rD,d(rA)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	43						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	d															

Definition Load Half Word Algebraic with Update

Operation

EA ← (**r**A) + EXTS(d)
rD ← EXTS(MEM(EA, 2))
rA ← EA

Description

EA is the sum (**r**A) + d. The half word in memory addressed by EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are filled with a copy of the most-significant bit of the loaded half word. EA is placed into rA. If **r**A = 0 or **r**A = **r**D, the instruction form is invalid.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

lhaux

Assembler Syntax lhaux rD,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						375									0

Definition Load Half Word Algebraic with Update Indexed

Operation

EA ← (**r**A) + (**r**B)
rD ← EXTS(MEM(EA, 2))
rA ← EA

Description

EA is the sum (**r**A) + (**r**B). The half word in memory addressed by EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are filled with a copy of the most-significant bit of the loaded half word. EA is placed into **r**A. If rA = 0 or rA = rD, the instruction form is invalid.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

lhax

Assembler Syntax lhax rD,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						343									0

Definition Load Half Word Algebraic Indexed

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + (**r**B)
rD ← EXTS(MEM(EA, 2))

Description EA is the sum (**r**AI0) + (**r**B). The half word in memory addressed by EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are filled with a copy of the most-significant bit of the loaded half word.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

lhbrx

Assembler Syntax lhbrx rD,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						790									0

Definition Load Half Word Byte-Reverse Indexed

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + (**r**B)
rD ← (16)0 || MEM(EA + 1, 1) || MEM(EA, 1)

Description

EA is the sum (**r**A|0) + (**r**B). Bits 0–7 of the half word in memory addressed by EA are loaded into the low-order eight bits of **r**D. Bits 8–15 of the half word in memory addressed by EA are loaded into the subsequent low-order eight bits of **r**D. The remaining bits in **r**D are cleared.

The PowerPC architecture cautions programmers that some implementations of the architecture may run the **lhbrx** instructions with greater latency than other types of load instructions.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

lhz

Assembler Syntax lhz rD,d(rA)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	40						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	d															

Definition Load Half Word and Zero

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + EXTS(d)
rD ← (16)0 || MEM(EA, 2)

Description EA is the sum (**r**A|0) + d. The half word in memory addressed by EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are cleared.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

lhzu

Assembler Syntax lhzu rD,d(rA)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	41						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	d															

Definition Load Half Word and Zero with Update

Operation

EA ← **r**A + EXTS(d)
rD ← (16)0 || MEM(EA, 2)
rA ← EA

Description EA is the sum (**r**A) + d. The half word in memory addressed by EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are cleared. EA is placed into **r**A. If **r**A = 0 or **r**A = **r**D, the instruction form is invalid.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

lhzux

Assembler Syntax lhzux rD,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					311										0

Definition Load Half Word and Zero with Update Indexed

Operation

EA ← (**r**A) + (**r**B)
rD ← (16)0 || MEM(EA, 2)
rA ← EA

Description

EA is the sum (**r**A) + (**r**B). The half word in memory addressed by EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are cleared. EA is placed into **r**A. If **r**A = 0 or **r**A = **r**D, the instruction form is invalid.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

lhzx

Assembler Syntax lhzx rD,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						279									0

Definition Load Half Word and Zero Indexed

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + (**r**B)
rD ← (16)0 || MEM(EA, 2)

Description EA is the sum (**r**A|0) + (**r**B). The half word in memory addressed by EA is loaded into the low-order 16 bits of **r**D. The remaining bits in **r**D are cleared.

Other registers altered:

- None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

lmw

Assembler Syntax lmw rD,d(rA)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	46						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	d															

Definition Load Multiple Word

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + EXTS(d)
r ← **r**D
do while r ≤ 31
GPR(r) ← MEM(EA, 4)
r ← r + 1
EA ← EA + 4

Description

EA is the sum (**r**A|0) + d. *n* = (32 – **r**D). *n* consecutive words starting at EA are loaded into GPRs **r**D through **r31**.

EA must be a multiple of four. If it is not, either the system alignment exception handler is invoked or the results are boundedly undefined. If **r**A is in the range of registers specified to be loaded, including the case in which **r**A = 0, the instruction form is invalid.

Note that, in some implementations, this instruction is likely to have a greater latency and take longer to execute, perhaps much longer, than a sequence of individual load or store instructions that produce the same results.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

lswi

Assembler Syntax lswi rD,rA,NB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	NB						597									0

Definition Load String Word Immediate

Operation

if **r**A = 0 then EA ← 0
else EA ← (**r**A)
if NB = 0 then *n* ← 32
else *n* ← NB
r ← **r**D – 1
i ← 32
do while *n* > 0
if i = 32 then
r ← r + 1 (mod 32)
GPR(r) ← 0
GPR(r)[i–i + 7] ← MEM(EA, 1)
i ← i + 8
if i = 32 then i ← 0
EA ← EA + 1
n ← *n* – 1

Description

EA is (rAI0). Let *n* = NB if NB ≠ 0, *n* = 32 if NB = 0; *n* is the number of bytes to load. Let *nr* = CEIL(*n* ÷ 4); *nr* is the number of registers to be loaded with data.

n consecutive bytes starting at EA are loaded into GPRs **r**D through **r**D + *nr* – 1. Bytes are loaded left to right in each register. The sequence of registers wraps around to **r0** if required. If the 4 bytes of register **r**D + *nr* – 1 are only partially filled, the unfilled low-order byte(s) of that register are cleared. If **r**A is in the range of registers specified to be loaded, including the case in which **r**A = 0, the instruction form is invalid. Under certain conditions (for example, segment boundary crossing) the data alignment exception handler may be invoked.

Note that, in some implementations, this instruction is likely to have greater latency and take longer to execute, perhaps much longer, than a sequence of individual load or store instructions that produce the same results.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

lswx

Assembler Syntax lswx rD,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						533									0

Definition Load String Word Indexed

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + (**r**B)
n ← XER[25–31]
r ← **r**D – 1
i ← 32
rD ← undefined
do while *n* > 0
if i = 32 then
r ← r + 1 (mod 32)
GPR(r) ← 0
GPR(r)[i–i + 7] ← MEM(EA, 1)
i ← i + 8
if i = 32 then i ← 0
EA ← EA + 1
n ← *n* – 1

Description

EA is the sum (**r**AI0) + (**r**B). Let *n* = XER[25–31]; *n* is the number of bytes to load. Let *nr* = CEIL(*n* ÷ 4); *nr* is the number of registers to receive data. If *n* > 0, *n* consecutive bytes starting at EA are loaded into GPRs **r**D through **r**D + *nr* – 1.

Bytes are loaded left to right in each register. The sequence of registers wraps around through **r0** if required. If the four bytes of **r**D + *nr* – 1 are only partially filled, the unfilled low-order byte(s) of that register are cleared. If *n* = 0, the contents of **r**D are undefined.

If **r**A or **r**B is in the range of registers specified to be loaded, including the case in which **r**A = 0, either the system illegal instruction error handler is invoked or the results are boundedly undefined. If **r**D = **r**A or **r**D = **r**B, the instruction form is invalid. If **r**D and **r**A both specify GPR0, the form is invalid.

Under certain conditions (for example, segment boundary crossing) the data alignment exception handler may be invoked. Note that, in some implementations, this instruction is likely to have a greater latency and take longer to execute, perhaps much longer, than a sequence of individual load or store instructions that produce the same results.

Other registers altered:

- None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

lwarx

Assembly Syntax lwarx rD,rA,rB

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

Definition Load Word and Reserve Indexed

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + (**r**B)
RESERVE ← 1
RESERVE_ADDR ← physical_addr(EA)
rD ← MEM(EA,4)

Description

EA is the sum (**r**A|0) + (**r**B). The word in memory addressed by EA is loaded into **r**D.

This instruction creates a reservation for use by a store word conditional indexed (**stwcx.**) instruction. The physical address computed from EA is associated with the reservation, and replaces any address previously associated with the reservation. EA must be a multiple of four. If it is not, either the system alignment exception handler is invoked or the results are boundedly undefined. When the RESERVE bit is set, the processor enables hardware snooping for the block of memory addressed by the RESERVE address.

If the processor detects that another processor writes to the block of memory it has reserved, it clears the RESERVE bit. The **stwcx.** instruction will only do a store if the RESERVE bit is set. The **stwcx.** instruction sets the CR0[EQ] bit if the store was successful and clears it if it failed. The **lwarx** and **stwcx.** combination can be used for atomic read-modify-write sequences.

Note that the atomic sequence is not guaranteed, but its failure can be detected if CR0[EQ] = 0 after the **stwcx.** instruction.

Other registers altered:

- None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

lwbrx

Assembler Syntax lwbrx rD,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						534									0

Definition Load Word Byte-Reverse Indexed

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + (**r**B)
rD ← MEM(EA + 3, 1) || MEM(EA + 2, 1) || MEM(EA + 1, 1) || MEM(EA, 1)

Description

EA is the sum (**r**A|0) + **r**B. Bits 0–7 of the word in memory addressed by EA are loaded into the low-order 8 bits of **r**D. Bits 8–15 of the word in memory addressed by EA are loaded into the subsequent low-order 8 bits of **r**D. Bits 16–23 of the word in memory addressed by EA are loaded into the subsequent low-order eight bits of **r**D. Bits 24–31 of the word in memory addressed by EA are loaded into the subsequent low-order 8 bits of **r**D. The MPC823 may run the **lwbrx** instructions with greater latency than other types of load instructions.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

lwz

Assembler Syntax lwz rD,d(rA)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	32						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	d															

Definition Load Word and Zero

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + EXTS(d)
rD ← MEM(EA, 4)

Description EA is the sum (**r**A|0) + d. The word in memory addressed by EA is loaded into **r**D.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

lwzu

Assembler Syntax lwzu rD,d(rA)

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

Definition Load Word and Zero with Update

Operation

EA ← **r**A + EXTS(d)
rD ← MEM(EA, 4)
rA ← EA

Description EA is the sum (**r**A) + d. The word in memory addressed by EA is loaded into **r**D. EA is placed into **r**A. If **r**A = 0, or **r**A = **r**D, the instruction form is invalid.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

lwzux

Assembler Syntax lwzux rD,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					55										0

Definition Load Word and Zero with Update Indexed

Operation

EA ← (**r**A) + (**r**B)
rD ← MEM(EA, 4)
rA ← EA

Description EA is the sum (**r**A) + (**r**B). The word in memory addressed by EA is loaded into **r**D. EA is placed into **r**A. If **r**A = 0, or **r**A = **r**D, the instruction form is invalid.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

lwzx

Assembler Syntax lwzx rD,rA,rB

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

Definition Load Word and Zero Indexed

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + **r**B
rD ← MEM(EA, 4)

Description EA is the sum (**r**A|0) + (**r**B). The word in memory addressed by EA is loaded into **r**D.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

mcrf

Assembler Syntax mcrf crfD,crfS

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	19						CRFD			00		CRFS			00	
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000						0000000000									0

Definition Move Condition Register Field

Operation CR[4 * **crf**D–4 * **crf**D + 3] ← CR[4 * **crf**S–4 * **crf**S + 3]

Description The contents of condition register field **crf**S are copied into condition register field **crf**D. All other condition register fields remain unchanged.

Other registers altered:

- Condition Register (CR field specified by operand crfD): Affected: LT, GT, EQ, SO

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XL

mcrxr

Assembler Syntax mcrxr crfD

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						CRFD			00		00000				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000						512									0

Definition Move to Condition Register from XER

Operation CR[4 * **crf**D–4 * **crf**D + 3] ← XER[0–3]
XER[0–3] ← 0b0000

Description The contents of XER[0–3] are copied into the condition register field designated by **crf**D. All other fields of the condition register remain unchanged. XER[0–3] is cleared.

Other registers altered:

- Condition Register (CR field specified by operand crfD): Affected: LT, GT, EQ, SO
- XER[0–3]

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

mfcr

Assembler Syntax mfcr rD

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					00000				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000					19										0

Definition Move from Condition Register

Operation **r**D ← CR

Description The contents of the condition register (CR) are placed into **r**D.

Other registers altered:

- None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

mfmsr

Assembler Syntax mfmsr rD

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					00000				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000						83									0

Definition Move from Machine State Register

Operation **r**D ← MSR

Description The contents of the MSR are placed into **r**D. This is a supervisor level instruction.

Other registers altered:

- None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
OEA	√		X

mfspr

Assembler Syntax mfspr rD,SPR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					SPR*				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	SPR*						339									0

NOTE: *This is a split field.

Definition Move from Special-Purpose Register

Operation

$n \leftarrow$ spr[5–9] || spr[0–4]
rD $\leftarrow$ SPR(n)

Description In the PowerPC UISA, the SPR field denotes a special-purpose register, encoded as shown in the table below. The contents of the designated special-purpose register are placed into **r**D.

SPR*			REGISTER NAME
DECIMAL	SPR[5–9]	SPR[0–4]	
1	00000	00001	XER
8	00000	01000	LR
9	00000	01001	CTR

NOTE: *The order of the two 5-bit halves of the SPR number is reversed compared with the actual instruction coding.

If the SPR field contains any value other than one of the values shown in Table 9 (and the processor is in user mode), one of the following occurs:

- The system illegal instruction error handler is invoked.
- The system supervisor-level instruction error handler is invoked.
- The results are boundedly undefined.

Other registers altered:

- None

Simplified mnemonics:

mfxer	**r**D	equivalent to	**mfspr**	**r**D,**1**
mflr	**r**D	equivalent to	**mfspr**	**r**D,**8**
mfctr	**r**D	equivalent to	**mfspr**	**r**D,**9**

In the PowerPC OEA, the SPR field denotes a special-purpose register, encoded as shown in the table below. The contents of the designated SPR are placed into **r**D. SPR[0] = 1 if and only if reading the register is supervisor-level. Execution of this instruction specifying a defined and supervisor-level register when MSR[PR] = 1 will result in a priviledged instruction type program exception.

If MSR[PR] = 1, the only effect of executing an instruction with an SPR number that is not shown in the table below and has SPR[0] = 1 is to cause a supervisor-level instruction type program exception or an illegal instruction type program exception. For all other cases, MSR[PR] = 0 or SPR[0] = 0. If the SPR field contains any value that is not shown in the table, either an illegal instruction type program exception occurs or the results are boundedly undefined.

Other registers altered:

- None

SPR[1]			REGISTER NAME	ACCESS
DECIMAL	SPR[5–9]	SPR[0–4]		
1	00000	00001	XER	User
8	00000	01000	LR	User
9	00000	01001	CTR	User
18	00000	10010	DSISR	Supervisor
19	00000	10011	DAR	Supervisor
22	00000	10110	DEC	Supervisor
26	00000	11010	SRR0	Supervisor
27	00000	11011	SRR1	Supervisor
80	00010	10000	EIE[2]	Supervisor
81	00010	10001	EID[3]	Supervisor
144	00100	10000	CMPA[4]	Supervisor
145	00100	10001	CMPB[4]	Supervisor

SPR[1]			REGISTER NAME	ACCESS
DECIMAL	SPR[5–9]	SPR[0–4]		
146	00100	10010	CMPC[4]	Supervisor
147	00100	10011	CMPD[4]	Supervisor
148	00100	10100	ICR[4]	Supervisor
149	00100	10101	DER[4]	Supervisor
150	00100	10110	COUNTA[4]	Supervisor
151	00100	10111	COUNTB[4]	Supervisor
152	00100	11000	CMPE[4]	Supervisor
153	00100	11001	CMPF[4]	Supervisor
154	00100	11010	CMPG[4]	Supervisor
155	00100	11011	CMPH[4]	Supervisor
156	00100	11100	LCTRL1[4]	Supervisor
157	00100	11101	LCTRL2[4]	Supervisor
158	00100	11110	ICTRL[4]	Supervisor
159	00100	11111	BAR[4]	Supervisor
272	01000	10000	SPRG0	Supervisor
273	01000	10001	SPRG1	Supervisor
274	01000	10010	SPRG2	Supervisor
275	01000	10011	SPRG3	Supervisor
287	01000	11111	PVR	Supervisor
560	10001	10000	IC_CST	Supervisor
561	10001	10001	IC_ADR	Supervisor
562	10001	10010	IC_DAT	Supervisor
568	10001	11000	DC_CST	Supervisor
569	10001	11001	DC_ADR	Supervisor
570	10001	11010	DC_DAT	Supervisor
630	10011	10110	DPDR[4]	Supervisor
638	10011	11110	IMMR	Supervisor
784	11000	10000	MI_CTR	Supervisor
786	11000	10010	MI_AP	Supervisor
787	11000	10011	MI_EPN	Supervisor
789	11000	10101	MI_TWC	Supervisor
790	11000	10110	MI_RPN	Supervisor
792	11000	11000	MD_CTR	Supervisor
793	11000	11001	M_CASID	Supervisor

SPR[1]			REGISTER NAME	ACCESS
DECIMAL	SPR[5–9]	SPR[0–4]		
794	11000	11010	MD_AP	Supervisor
795	11000	11011	MD_EPN	Supervisor
796	11000	11100	M_TWB	Supervisor
797	11000	11101	MD_TWC	Supervisor
798	11000	11110	MD_RPN	Supervisor
799	11000	11111	M_TW	Supervisor
816	11001	10000	MI_DBCAM	Supervisor
817	11001	10001	MI_DBRAM0	Supervisor
818	11001	10010	MI_DBRAM1	Supervisor
824	11001	11000	MD_DBCAM	Supervisor
825	11001	11001	MI_DBRAM0	Supervisor
826	11001	11010	MI_DBRAM1	Supervisor

NOTES:

1. The order of the two 5-bit halves of the SPR number is reversed compared with actual instruction coding.
2. Sets the EE Bit in the MSR.
3. Clears the EE Bit in the MSR.
4. Development Support (Debug) Register.

   For **mtspr** and **mfspr** instructions, the SPR number coded in assembly language does not appear directly as a 10-bit binary number in the instruction. The number coded is split into two 5-bit halves that are reversed in the instruction, with the high-order five bits appearing in bits 16–20 of the instruction and the low-order five bits in bits 11–15.

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA/OEA	√		XFX

NOTE: **mfspr** is supervisor-level only if SPR[0] = 1.

mftb

Assembler Syntax mftb rD,TBR

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					TBR				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	TBR						371									0

Definition Move from Time Base

Operation

$n \leftarrow$ tbr[5–9] || tbr[0–4]
if n = 268 then
rD $\leftarrow$ TBL
else if n = 269 then
rD $\leftarrow$ TBU

TBR*			REGISTER NAME	ACCESS
DECIMAL	TBR[5–9]	TBR[0–4]		
268	01000	01100	TB Read	User
269	01000	01101	TBU Read	User

NOTE: *The order of the two 5-bit halves of the TBR number is reversed.

Description

If the TBR field contains any value other than one of the values shown in the table above, then one of the following occurs:

- The system illegal instruction error handler is invoked.
- The system supervisor-level instruction error handler is invoked.
- The results are boundedly undefined.

It is important to note that some implementations may implement **mftb** and **mfspr** identically, therefore, a TBR number must not match an SPR number.

Other registers altered:

- None

Simplified mnemonics:

mftb	**r**D	equivalent to	**mftb**	**r**D,**268**
mftbu	**r**D	equivalent to	**mftb**	**r**D,**269**

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
VEA			XFX

mtcrf

Assembler Syntax mtcrf CRM,rS

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					0	CRM			
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	CRM				0	144										0

Definition Move to Condition Register Fields

Operation

mask ← (4)(CRM[0]) || (4)(CRM[1]) ||... (4)(CRM[7])
CR ← (**r**S & mask) | (CR & ¬ mask)

Description

The contents of **r**S are placed into the condition register under control of the field mask specified by CRM. The field mask identifies the 4-bit fields affected. Let i be an integer in the range 0–7. If CRM(i) = 1, CR field i (CR bits 4 * i through 4 * i + 3) is set to the contents of the corresponding field of **r**S.

Note that updating a subset of the eight fields of the condition register may have substantially poorer performance on some implementations than updating all of the fields.

Other registers altered:

- CR fields selected by mask

Simplified mnemonics:

mtcr **r**S equivalent to **mtcrf** 0xFF,**r**S

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XFX

mtmsr

Assembler Syntax mtmsr rS

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					00000				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000						146									0

Definition Move to Machine State Register

Operation MSR ← (**r**S)

Description The contents of **r**S are placed into the MSR. This is a supervisor level instruction. It is also an execution synchronizing instruction except with respect to alterations to the POW and LE bits.

In addition, alterations to the MSR[EE] and MSR[RI] bits are effective as soon as the instruction completes. Thus if MSR[EE] = 0 and an external or decrementer exception is pending, executing an **mtmsr** instruction that sets MSR[EE] = 1 will cause the external or decrementer exception to be taken before the next instruction is executed, if no higher priority exception exists.

Other registers altered:

- MSR

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
OEA	√		X

mtspr

Assembler Syntax mtspr SPR,rS

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					SPR*				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	SPR*						467									0

NOTE: *This is a split field.

Definition Move to Special-Purpose Register

Operation

$n \leftarrow$ spr[5–9] || spr[0–4]
SPR(n) $\leftarrow$ **r**S

Description In the PowerPC UISA, the SPR field denotes a special-purpose register, encoded as shown in the following table. The contents of **r**S are placed into the designated special-purpose register.

SPR*			REGISTER NAME
DECIMAL	SPR[5–9]	SPR[0–4]	
1	00000	00001	XER
8	00000	01000	LR
9	00000	01001	CTR

NOTE: *The order of the two 5-bit halves of the SPR number is reversed compared with actual instruction coding.

If the SPR field contains any value other than one of the values shown in the table above, and the processor is operating in user mode, one of the following occurs:

- The system illegal instruction error handler is invoked.
- The system supervisor instruction error handler is invoked.
- The results are boundedly undefined.

Other registers altered:

- See table above.

Simplified mnemonics:

mtxer	**r**D	equivalent to	**mtspr**	**1,r**D
mtlr	**r**D	equivalent to	**mtspr**	**8,r**D
mtctr	**r**D	equivalent to	**mtspr**	**9,r**D

In the PowerPC OEA, the SPR field denotes a special-purpose register, encoded as shown in the following table. The contents of **r**S are placed into the designated special-purpose register. For this instruction, SPRs TBL and TBU are treated as separate 32-bit registers; setting one leaves the other unaltered.

The value of SPR[0] = 1 if and only if writing the register is a supervisor-level operation. Execution of this instruction specifying a defined and supervisor-level register when MSR[PR] = 1 results in a privileged instruction type program exception.

If MSR[PR] = 1 then the only effect of executing an instruction with an SPR number that is not shown in the following table and has SPR[0] = 1 is to cause a privileged instruction type program exception or an illegal instruction type program exception. For all other cases, MSR[PR] = 0 or SPR[0] = 0, if the SPR field contains any value that is not shown in the table, either an illegal instruction type program exception occurs or the results are boundedly undefined.

Other registers altered:

SPR[1]			REGISTER NAME	ACCESS
DECIMAL	SPR[5–9]	SPR[0–4]		
1	00000	00001	XER	User
8	00000	01000	LR	User
9	00000	01001	CTR	User
18	00000	10010	DSISR	Supervisor
19	00000	10011	DAR	Supervisor
22	00000	10110	DEC	Supervisor
26	00000	11010	SRR0	Supervisor
27	00000	11011	SRR1	Supervisor
80	00010	10000	EIE[2]	Supervisor
81	00010	10001	EID[3]	Supervisor

SPR[1]			REGISTER NAME	ACCESS
DECIMAL	SPR[5–9]	SPR[0–4]		
144	00100	10000	CMPA[4]	Supervisor
145	00100	10001	CMPB[4]	Supervisor
146	00100	10010	CMPC[4]	Supervisor
147	00100	10011	CMPD[4]	Supervisor
148	00100	10100	ICR[4]	Supervisor
149	00100	10101	DER[4]	Supervisor
150	00100	10110	COUNTA[4]	Supervisor
151	00100	10111	COUNTB[4]	Supervisor
152	00100	11000	CMPE[4]	Supervisor
153	00100	11001	CMPF[4]	Supervisor
154	00100	11010	CMPG[4]	Supervisor
155	00100	11011	CMPH[4]	Supervisor
156	00100	11100	LCTRL1[4]	Supervisor
157	00100	11101	LCTRL2[4]	Supervisor
158	00100	11110	ICTRL[4]	Supervisor
159	00100	11111	BAR[4]	Supervisor
272	01000	10000	SPRG0	Supervisor
273	01000	10001	SPRG1	Supervisor
274	01000	10010	SPRG2	Supervisor
275	01000	10011	SPRG3	Supervisor
284	01000	11100	TB Write	Supervisor
285	01000	11101	TBU Write	Supervisor
560	10001	10000	IC_CST	Supervisor
561	10001	10001	IC_ADR	Supervisor
562	10001	10010	IC_DAT	Supervisor
568	10001	11000	DC_CST	Supervisor
569	10001	11001	DC_ADR	Supervisor
570	10001	11010	DC_DAT	Supervisor
630	10011	10110	DPDR[4]	Supervisor
638	10011	11110	IMMR	Supervisor
784	11000	10000	MI_CTR	Supervisor
786	11000	10010	MI_AP	Supervisor
787	11000	10011	MI_EPN	Supervisor
789	11000	10101	MI_TWC	Supervisor

SPR[1]			REGISTER NAME	ACCESS
DECIMAL	SPR[5–9]	SPR[0–4]		
790	11000	10110	MI_RPN	Supervisor
792	11000	11000	MD_CTR	Supervisor
793	11000	11001	M_CASID	Supervisor
794	11000	11010	MD_AP	Supervisor
795	11000	11011	MD_EPN	Supervisor
796	11000	11100	M_TWB	Supervisor
797	11000	11101	MD_TWC	Supervisor
798	11000	11110	MD_RPN	Supervisor
799	11000	11111	M_TW	Supervisor
816	11001	10000	MI_DBCAM	Supervisor
817	11001	10001	MI_DBRAM0	Supervisor
818	11001	10010	MI_DBRAM1	Supervisor
824	11001	11000	MD_DBCAM	Supervisor
825	11001	11001	MI_DBRAM0	Supervisor
826	11001	11010	MI_DBRAM1	Supervisor

NOTES:

1. The order of the two 5-bit halves of the SPR number is reversed. For **mtspr** and **mfspr** instructions, the SPR number coded in assembly language does not appear directly as a 10-bit binary number in the instruction. The number coded is split into two 5-bit halves that are reversed in the instruction, with the high-order five bits appearing in bits 16–20 of the instruction and the low-order five bits in bits 11–15..
2. Sets EE Bit in MSR.
3. Clears EE Bit in MSR.
4. Development Support (Debug) Register.

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA/OEA	√		XFX

NOTE: **mtspr** is supervisor-level only if SPR[0] = 1.

mulhw

Assembler Syntax

mulhw rD,rA,rB (Rc = 0)
mulhw. rD,rA,rB (Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					0	75									RC

Definition

Multiply High Word

Operation

prod[0–63] ← **r**A * **r**B
rD ← prod[0–31]

Description

The 64 -bit product is formed from the contents of **r**A and **r**B. The high-order 32 bits of the 64-bit product of the operands are placed into **r**D. Both the operands and the product are interpreted as signed integers. This instruction may execute faster on some implementations if **r**B contains the operand having the smaller absolute value.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XO

mulhwu

Assembler Syntax

mulhwu rD,rA,rB(Rc = 0)
mulhwu. rD,rA,rB(Rc = 1)

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

Definition

Multiply High Word Unsigned

Operation

prod[0–63] ← **r**A * **r**B
rD ← prod[0–31]

Description

The 32-bit operands are the contents of **r**A and **r**B. The high order 32 bits of the 64-bit product of the operands are placed into **r**D. Both the operands and the product are interpreted as unsigned integers, except that if Rc = 1 the first three bits of CR0 field are set by signed comparison of the result to zero. This instruction may execute faster on some implementations if **r**B contains the operand having the smaller absolute value.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XO

mulli

Assembler Syntax mulli rD,rA,SIMM

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	07						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	SIMM															

Definition Multiply Low Immediate

Operation

prod[0–48] ← (**r**A) * SIMM
rD ← prod[16-48]

Description

The first operand is (**r**A). The 16-bit second operand is the value of the SIMM field. The low-order 32-bits of the 48-bit product of the operands are placed into **r**D. Both the operands and the product are interpreted as signed integers. The low-order 32 bits of the product are calculated independently of whether the operands are treated as signed or unsigned 32-bit integers. This instruction can be used with **mulhd***x* or **mulhw***x* to calculate a full 64-bit product.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

mullw

Assembler Syntax

mullw	rD,rA,rB (OE = 0 Rc = 0)
mullw.	rD,rA,rB (OE = 0 Rc = 1)
mullwo	rD,rA,rB (OE = 1 Rc = 0)
mullwo.	rD,rA,rB (OE = 1 Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					OE	235									RC

Definition

Multiply Low Word

Operation

rD ← **r**A * **r**B

Description

The 32-bit operands are the contents of **r**A and **r**B. The low-order 32 bits of the 64-bit product (**r**A) * (**r**B) are placed into **r**D. The low-order 32 bits of the product are the correct 32-bit product for 32-bit implementations. The low-order 32-bits of the product are independent of whether the operands are regarded as signed or unsigned 32-bit integers. If OE = 1, then OV is set if the product cannot be represented in 32 bits. Both the operands and the product are interpreted as signed integers.

Note that this instruction may execute faster on some implementations if **r**B contains the operand having the smaller absolute value.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1).
  The CR0 field may not reflect the "true" (infinitely precise) result if overflow occurs (see XER below).
- XER:
  Affected: SO, OV (if OE = 1).
  The setting of the affected bits in the XER is mode-independent, and reflects overflow of the 32-bit result.

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XO

nand

Assembler Syntax

nand rA,rS,rB (Rc = 0)
nand. rA,rS,rB (Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					476										RC

Definition

NAND

Operation

rA ← ¬ ((**r**S) & (**r**B))

Description

The contents of **r**S are ANDed with the contents of **r**B and the complemented result is placed into **r**A. **nand** with **r**S = **r**B can be used to obtain the one's complement.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

neg

Assembler Syntax

neg	rD,rA	(OE = 0 Rc = 0)
neg.	rD,rA	(OE = 0 Rc = 1)
nego	rD,rA	(OE = 1 Rc = 0)
nego.	rD,rA	(OE = 1 Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000					OE	104									RC

Definition

Negate

Operation

rD ← ¬ (**r**A) + 1

Description

The value 1 is added to the complement of the value in **r**A, and the resulting two's complement is placed into **r**D. If **r**A contains the most negative 32-bit number (0x8000_0000), the result is the most negative number and, if OE = 1, OV is set.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
- XER:
  Affected: SO OV (if OE = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XO

nor

Assembler Syntax

nor	rA,rS,rB (Rc = 0)
nor.	rA,rS,rB (Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					124										RC

Definition NOR

Operation **r**A ← ¬ ((**r**S) | (**r**B))

Description The contents of **r**S are ORed with the contents of **r**B and the complemented result is placed into **r**A. **nor** with **r**S = **r**B can be used to obtain the one's complement.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)

Simplified mnemonics:

not	**r**D,**r**S	equivalent to	**nor**	**r**A,**r**S,**r**S

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

or

Assembler Syntax

or	rA,rS,rB (Rc = 0)
or.	rA,rS,rB (Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						444									RC

Definition

OR

Operation

rA ← (**r**S) | (**r**B)

Description

The contents of **r**S are ORed with the contents of **r**B and the result is placed into **r**A. The simplified mnemonic **mr** (shown below) demonstrates the use of the **or** instruction to move register contents.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)

Simplified mnemonics:

mr	**r**A,**r**S	equivalent to	**or**	**r**A,**r**S,**r**S

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

orc

Assembler Syntax

orc	rA,rS,rB (Rc = 0)
orc.	rA,rS,rB (Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					412										RC

Definition OR with Complement

Operation **r**A ← (**r**S) | ¬ (**r**B)

Description The contents of **r**S are ORed with the complement of the contents of **r**B and the result is placed into **r**A.

Other registers altered:

- ❑ Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

ori

Assembler Syntax ori rA,rS,UIMM

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

Definition OR Immediate

Operation **r**A ← (**r**S) | ((16)0 || UIMM)

Description The contents of **r**S are ORed with 0x0000|| UIMM and the result is placed into **r**A. The preferred no-op (an instruction that does nothing) is **ori 0,0,0**.

Other registers altered:

- None

Simplified mnemonics:

nop		equivalent to	**ori**	**0,0,0**

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

oris

Assembler Syntax oris rA,rS,UIMM

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

Definition OR Immediate Shifted

Operation **r**A ← (**r**S) | (UIMM || (16)0)

Description The contents of **r**S are ORed with UIMM || 0x0000 and the result is placed into **r**A.

Other registers altered:

- None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

rfi

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	19						00000					00000				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000						50									0

Definition Return from Interrupt

Operation MSR[16–23, 25–27, 30–31] ← SRR1[16–23, 25–27, 30–31]
NIA ←iea SRR0[0–29] || 0b00

Description Bits SRR1[0,5-9,16-31] are placed into the corresponding bits of the MSR. If the new MSR value does not enable any pending exceptions, then the next instruction is fetched, under control of the new MSR value, from the address SRR0[0–29] || 0b00. If the new MSR value enables one or more pending exceptions, the exception associated with the highest priority pending exception is generated; in this case the value placed into SRR0 by the exception processing mechanism is the address of the instruction that would have been executed next had the exception not occurred. Note that an implementation may define addtional MSR bits, and in this case, may also cause them to be saved to SRR1 from MSR on an exception and restored to MSR from SRR1 on an **rfi.** This is a supervisor-level, context synchronizing instruction.

Other registers altered:

- ❑ MSR

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
OEA	√		XL

rlwimi

Assembler Syntax

rlwimi	rA,rS,SH,MB,ME (Rc = 0)
rlwimi.	rA,rS,SH,MB,ME (Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	20						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	SH					MB					ME					RC

Definition

Rotate Left Word Immediate then Mask Insert

Operation

$n \leftarrow$ SH
r $\leftarrow$ ROTL(**r**S, n)
m $\leftarrow$ MASK(MB, ME)
rA $\leftarrow$ (r & m) | (**r**A & ¬ m)

Description

The contents of **r**S are rotated left the number of bits specified by operand SH. A mask is generated having 1 bits from bit MB through bit ME and 0 bits elsewhere. The rotated data is inserted into **r**A under control of the generated mask.

Note that rlwimi can be used to insert a bit field into the contents of rA using the methods shown below:

- To insert an n-bit field, that is left-justified rS, into rA starting at bit position b, set SH = 32 – b, MB = b, and ME = (b + n) – 1.
- To insert an n-bit field, that is right-justified in rS, into rA starting at bit position b, set SH = 32 – (b + n), MB = b, and ME = (b + n) – 1.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO(if Rc = 1)

Simplified mnemonics:

inslwi	**r**A,**r**S,*n*,*b*	equivalent to	**rlwimi**	**r**A,**r**S,32 – *b*,*b*,*b* + *n* – 1
insrwi	**r**A,**r**S,*n*,*b* (n > 0)	equivalent to	**rlwimi**	**r**A,**r**S,**32** – (*b* + *n*),*b*,(*b* + *n*) – 1

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			M

rlwinm

Assembler Syntax

rlwinm rA,rS,SH,MB,ME (Rc = 0)
rlwinm. rA,rS,SH,MB,ME (Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	21						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	SH					MB						ME				RC

Definition

Rotate Left Word Immediate then AND with Mask

Operation

$n \leftarrow$ SH
r $\leftarrow$ ROTL(**r**S, n)
m $\leftarrow$ MASK(MB, ME)
rA $\leftarrow$ r & m

Description

The contents of **r**S are rotated left the number of bits specified by operand SH. A mask is generated having 1 bits from bit MB through bit ME and 0 bits elsewhere. The rotated data is ANDed with the generated mask and the result is placed into **r**A.

Description

rlwinm can be used to extract, rotate, shift, and clear bit fields using the methods shown below:

- To extract an n field, that starts at bit position b in **r**S, right-justified into **r**A (clearing the remaining 32 – n bits of **r**A), set SH = b + n, MB = 32 – n, and ME = 31.
- To extract an n field, that starts at bit position b in rS, left-justified into rA (clearing the remaining 32 – n bits of **r**A), set SH = b, MB = 0, and ME = n – 1.
- To rotate the contents of a register left (or right) by n bits, set SH = n (32 – n), MB = 0, and ME = 31.
- To shift the contents of a register right by n bits, by setting SH = 32 – n, MB = n, and ME = 31. It can be used to clear the high-order b bits of a register and then shift the result left by n bits by setting SH = n, MB = b – n and ME = 31 – n.
- To clear the low-order n bits of a register, by setting SH = 0, MB = 0, and ME = 31 – n.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO(if Rc = 1)

Simplified mnemonics:

extlwi	**r**A,**r**S,*n*,*b* (*n* > 0)	equivalent to	**rlwinm**	**r**A,**r**S,b,**0**,*n* – 1
extrwi	**r**A,**r**S,*n*,*b* (*n* > 0)	equivalent to	**rlwinm**	**r**A,**r**S,b + *n*,32 – *n*,**31**
rotlwi	**r**A,**r**S,*n*	equivalent to	**rlwinm**	**r**A,**r**S,*n*,**0,31**
rotrwi	**r**A,**r**S,*n*	equivalent to	**rlwinm**	**r**A,**r**S,**32** – *n*,**0,31**
slwi	**r**A,**r**S,*n* (*n* < 32)	equivalent to	**rlwinm**	**r**A,**r**S,*n*,**0**,31–*n*
srwi	**r**A,**r**S,*n* (*n* < 32)	equivalent to	**rlwinm**	**r**A,**r**S,32 – *n*,*n*,**31**
clrlwi	**r**A,**r**S,*n* (*n* < 32)	equivalent to	**rlwinm**	**r**A,**r**S,**0**,*n*,**31**
clrrwi	**r**A,**r**S,*n* (*n* < 32)	equivalent to	**rlwinm**	**r**A,**r**S,**0,0**,31 – *n*
clrlslwi	**r**A,**r**S,*b*,*n* (*n* ≤ *b* < 32)	equivalent to	**rlwinm**	**r**A,**r**S,*n*,*b* – *n*,31 – *n*

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			M

rlwnm

Assembler Syntax

rlwnm rA,rS,rB,MB,ME (Rc = 0)
rlwnm. rA,rS,rB,MB,ME (Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	23						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					MB					ME					0

Definition

Rotate Left Word then AND with Mask

Operation

n ← **r**B[27-31]
r ← ROTL(**r**S, *n*)
m ← MASK(MB, ME)
rA ← r & m

Description

The contents of **r**S are rotated left the number of bits specified by the low-order five bits of **r**B. A mask is generated having 1 bits from bit MB through bit ME and 0 bits elsewhere. The rotated data is ANDed with the generated mask and the result is placed into **r**A.

rlwnm can be used to extract and rotate bit fields using the methods shown as follows:

- To extract an n field, that starts at variable bit position b in **r**S, right-justified into **r**A (clearing the remaining 32 – n bits of **r**A), by setting the low-order five bits of **r**B to b + n, MB = 32 – n, and ME = 31.
- To extract an n field, that starts at variable bit position b in **r**S, left-justified into **r**A (clearing the remaining 32 – n bits of **r**A), by setting the low-order five bits of **r**B to b, MB = 0, and ME = n – 1.
- To rotate the contents of a register left (or right) by n bits, by setting the low-order five bits of **r**B to n (32 – n), MB = 0, and ME = 31.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)

Simplified mnemonics:

rotlw	**rA,rS,rB**	equivalent to	**rlwnm**	**A,rS,rB,0,31**

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			M

sc

Assembler Syntax sc

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	17						00000					00000				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000000000000														1	0

Definition System Call

Description The **sc** instruction calls the operating system to perform a service. When control is returned to the program that executed the system call, the content of the registers depends on the register conventions used by the program providing the system service.

The effective address of the instruction following the **sc** instruction is placed into SRR0. Bits 0, 5-9, and 16-31 of the MSR are placed into the corresponding bits of SRR1, and bits 1-4 and 10-15 of SRR1 are set to undefined values. An **sc** exception is generated. The exception alters the MSR. The exception causes the next instruction to be fetched from offset 0xC00 from the base real address indicated by the new setting of MSR[IP].

Other registers altered:

- ❑ Dependent on the system service
- ❑ SRR0
- ❑ SRR1
- ❑ MSR

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA/OEA			SC

slw

Assembler Syntax

slw	rA,rS,rB (Rc = 0)
slw.	rA,rS,rB (Rc = 1)

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

Definition

Shift Left Word

Operation

$n \leftarrow$ **r**B[27-31]
rA $\leftarrow$ ROTL(**r**S, n)

Description

If bit 26 of **r**B = 0, the contents of **r**S are shifted left the number of bits specified by **r**B[27–31]. Bits shifted out of position 0 are lost. Zeros are supplied to the vacated positions on the right. The 32-bit result is placed into **r**A. If bit 26 of **r**B = 1, 32 zeros are placed into **r**A.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

sraw

Assembler Syntax

sraw	rA,rS,rB (Rc = 0)
sraw.	rA,rS,rB (Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					792										RC

Definition

Shift Right Algebraic Word

Operation

$n \leftarrow$ **r**B[27-31]
rA $\leftarrow$ ROTL(**r**S, n)

Description

If **r**B[26] = 0,then the contents of **r**S are shifted right the number of bits specified by **r**B[27–31]. Bits shifted out of position 31 are lost. The result is padded on the left with sign bits before being placed into **r**A. If **r**B[26] = 1, then **r**A is filled with 32 sign bits (bit 0) from **r**S. CR0 is set based on the value written into **r**A. XER[CA] is set if **r**S contains a negative number and any 1 bits are shifted out of position 31; otherwise XER[CA] is cleared. A shift amount of zero causes XER[CA] to be cleared.

The **sraw** instruction, followed by **addze**, can by used to divide quickly by 2^n. The setting of the XER[CA] bit, by **sraw**, is independent of mode.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO(if Rc = 1)
- XER:
  Affected: CA

srawi

Assembler Syntax

srawi	rA,rS,SH (Rc = 0)
srawi.	rA,rS,SH (Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	SH					824										RC

Definition

Shift Right Algebraic Word Immediate

Operation

$n \leftarrow$ SH
r $\leftarrow$ ROTL(**r**S, 32 – n)

Description

The contents of **r**S are shifted right the number of bits specified by operand SH. Bits shifted out of position 31 are lost. The shifted value is sign-extended before being placed in **r**A. The 32-bit result is placed into **r**A. XER[CA] is set if **r**S contains a negative number and any 1 bits are shifted out of position 31; otherwise XER[CA] is cleared. A shift amount of zero causes XER[CA] to be cleared.

The **srawi** instruction, followed by **addze**, can be used to divide quickly by 2^n. The setting of the CA bit, by **srawi**, is independent of mode.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
- XER:
  Affected: CA

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

srw

Assembler Syntax

srw	rA,rS,rB (Rc = 0)
srw.	rA,rS,rB (Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					536										RC

Definition

Shift Right Word

Operation

$n \leftarrow$ **r**B[27-31]
r $\leftarrow$ ROTL(**r**S, 32 $- n$)

Description

The contents of **r**S are shifted right the number of bits specified by the low-order six bits of **r**B. Bits shifted out of position 31 are lost. Zeros are supplied to the vacated positions on the left. The result is placed into **r**A.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

stb

Assembler Syntax stb rS,d(rA)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	38						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	d															

Definition Store Byte

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + EXTS(d)
MEM(EA, 1) ← **r**S[24-31]

Description

EA is the sum (**r**A|0) + d. The contents of the low-order eight bits of **r**S are stored into the byte in memory addressed by EA.

Other registers altered:

- None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

stbu

Assembler Syntax stbu rS,d(rA)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	39						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	d															

Definition Store Byte with Update

Operation

EA ← (**r**A) + EXTS(d)
MEM(EA, 1) ← **r**S[24-31]
rA ← EA

Description

EA is the sum (**r**A) + d. The contents of the low-order eight bits of **r**S are stored into the byte in memory addressed by EA. EA is placed into **r**A. If **r**A = 0, the instruction form is invalid.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

stbux

Assembler Syntax stbux rS,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					247										0

Definition Store Byte with Update Indexed

Operation

EA ← (**r**A) + (**r**B)
MEM(EA, 1) ← **r**S[24-31]
rA ← EA

Description

EA is the sum (**r**A) + (**r**B). The contents of the low-order eight bits of **r**S are stored into the byte in memory addressed by EA. EA is placed into **r**A. If **r**A = 0, the instruction form is invalid.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

stbx

Assembler Syntax stbx rS,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						215									0

Definition Store Byte Indexed

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + (**r**B)
MEM(EA, 1) ← **r**S[24-31]

Description

EA is the sum (**r**A|0) + (**r**B). The contents of the low-order eight bits of **r**S are stored into the byte in memory addressed by EA.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

sth

Assembler Syntax sth rS,d(rA)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	44						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	d															

Definition Store Half Word

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + EXTS(d)
MEM(EA, 2) ← **r**S[16-31]

Description

EA is the sum (**r**A|0) + d. The contents of the low-order 16 bits of **r**S are stored into the half word in memory addressed by EA.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

sthbrx

Assembler Syntax sthbrx rS,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						918									0

Definition Store Half Word Byte-Reverse Indexed

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + (**r**B)
MEM(EA, 2) ← **r**S[24-31] || **r**S[16-23]

Description

EA is the sum (**r**A|0) + (**r**B). The contents of the low-order eight bits of **r**S are stored into bits 0–7 of the half word in memory addressed by EA. The contents of the subsequent low-order eight bits of **r**S are stored into bits 8–15 of the half word in memory addressed by EA.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

sthu

Assembler Syntax sthu rS,d(rA)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	45						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	d															

Definition Store Half Word with Update

Operation

EA ← (**r**A) + EXTS(d)
MEM(EA, 2) ← **r**S[16-31]
rA ← EA

Description

EA is the sum (**r**A) + d. The contents of the low-order 16 bits of **r**S are stored into the half word in memory addressed by EA. EA is placed into **r**A. If **r**A = 0, the instruction form is invalid.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

sthux

Assembler Syntax sthux rS,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					439										0

Definition Store Half Word with Update Indexed

Operation

EA ← (**r**A) + (**r**B)
MEM(EA, 2) ← **r**S[16-31]
rA ← EA

Description

EA is the sum (**r**A) + (**r**B). The contents of the low-order 16 bits of **r**S are stored into the half word in memory addressed by EA. EA is placed into **r**A. If **r**A = 0, the instruction form is invalid.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

sthx

Assembler Syntax sthx rS,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					407										0

Definition Store Half Word Indexed

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + (**r**B)
MEM(EA, 2) ← **r**S[16-31]

Description

EA is the sum (**r**A|0) + (**r**B). The contents of the low-order 16 bits of **r**S are stored into the half word in memory addressed by EA.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

stmw

Assembler Syntax stmw rS,d(rA)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	47						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	d															

Definition Store Multiple Word

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + EXTS(d)
r ← **r**S
do while r ≤ 31
MEM(EA, 4) ← GPR(r)
r ← r + 1
EA ← EA + 4

Description

EA is the sum (**r**A|0) + d. *n* = (32 – **r**S). *n* consecutive words starting at EA are stored from the GPRs **r**S through **r31**. For example, if **r**S = 30, 2 words are stored. EA must be a multiple of four. If it is not, either the system alignment exception handler is invoked or the results are boundedly undefined.

Note that this instruction is likely to have a greater latency and take longer to execute, perhaps much longer, than a sequence of individual load or store instructions that produce the same results.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

stswi

Assembler Syntax stswi rS,rA,NB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	NB					725										0

Definition Store String Word Immediate

Operation

if **r**A = 0 then EA ← 0
else EA ← (**r**A)
if NB = 0 then *n* ← 32
else *n* ← NB
r ← **r**S – 1
i ← 32
do while *n* > 0
if i = 32 then r ← r + 1 (mod 32)
MEM(EA, 1) ← GPR(r)[i–i + 7]
i ← i + 8
if i = 64 then i ← 32
EA ← EA + 1
n ← *n* – 1

Description

EA is (**r**AI0). Let *n* = NB if NB ≠ 0, *n* = 32 if NB = 0; *n* is the number of bytes to store. Let *nr* = CEIL(*n* ÷ 4); *nr* is the number of registers to supply data. *n* consecutive bytes starting at EA are stored from GPRs **r**S through **r**S + *nr* – 1. Bytes are stored left to right from each register. The sequence of registers wraps around through **r0** if required. Under certain conditions (like segment boundary crossing), the data alignment exception handler may be invoked. In some implementations, this instruction is likely to have a greater latency and take longer to execute than a sequence of individual load or store instructions that produce the same results.

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

stswx

Assembler Syntax stswx **rS,rA,rB**

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						861									0

Definition Store String Word Indexed

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + (**r**B)
n ← XER[25–31]
r ← **r**S – 1
i ← 32
do while *n* > 0
if i = 32 then r ← r + 1 (mod 32)
MEM(EA, 1) ← GPR(r)[i–i + 7]
i ← i + 8
if i = 64 then i ← 32
EA ← EA + 1
n ← *n* – 1

Description

EA is the sum (**r**AI0) + (**r**B). Let *n* = XER[25–31]; *n* is the number of bytes to store. Let *nr* = CEIL(*n* ÷ 4); *nr* is the number of registers to supply data. *n* consecutive bytes starting at EA are stored from GPRs **r**S through **r**S + *nr* – 1. Bytes are stored left to right from each register. The sequence of registers wraps around through **r0** if required. If *n* = 0, no bytes are stored. Under certain conditions (for example, segment boundary crossing) the data alignment exception handler may be invoked.

Note that, in some implementations, this instruction is likely to have a greater latency and take longer to execute, perhaps much longer, than a sequence of individual load or store instructions that produce the same results.

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

stw

Assembler Syntax stw rS,d(rA)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	36						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	d															

Definition Store Word

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + EXTS(d)
MEM(EA, 4) ← **r**S

Description

EA is the sum (**r**A|0) + d. The contents of **r**S are stored into the word in memory addressed by EA.

Other registers altered:

- None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

stwbrx

Assembler Syntax stwbrx rS,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						662									0

Definition Store Word Byte-Reverse Indexed

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + (**r**B)
MEM(EA, 4) ← **r**S[24-31] || **r**S[16-23] || **r**S[8-15] || **r**S[0-7]

Description

EA is the sum (**r**A|0) + (**r**B). The contents of the low-order eight bits of **r**S are stored into bits 0–7 of the word in memory addressed by EA. The contents of the subsequent eight low-order bits of **r**S are stored into bits 8–15 of the word in memory addressed by EA. The contents of the subsequent eight low-order bits of **r**S are stored into bits 16–23 of the word in memory addressed by EA. The contents of the subsequent eight low-order bits of **r**S are stored into bits 24–31 of the word in memory addressed by EA.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

stwcx.

Assembler Syntax stwcx. rS,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						150									1

Definition Store Word Conditional Indexed

Operation

```
if rA = 0 then b ← 0
else b ← (rA)
EA ← b + (rB)
if RESERVE then
if RESERVE_ADDR = physical_addr(EA)
MEM(EA, 4) ← rS
CR0 ← 0b00 || 0b1 || XER[SO]
else
u ← undefined 1-bit value
if u then MEM(EA, 4) ← rS
CR0 ← 0b00 || u || XER[SO]
RESERVE ← 0
else
CR0 ← 0b00 || 0b0 || XER[SO]
```

Description

EA is the sum (rA|0) + (rB). If the reserved bit is set, the **stwcx.** instruction stores rS to effective address (rA + rB), clears the reserved bit, and sets CR0[EQ]. If the reserved bit is not set, the **stwcx.** instruction does not do a store; it leaves the reserved bit cleared and clears CR0[EQ]. Software must look at CR0[EQ] to see if the **stwcx.** was successful.

The reserved bit is set by the **lwarx** instruction. The reserved bit is cleared by any **stwcx.** instruction to any address, and also by snooping logic if it detects that another processor does any kind of store to the block indicated in the reservation buffer when reserved is set.

If a reservation exists, and the memory address specified by the **stwcx.** instruction is the same as that specified by the load and reserve instruction that established the reservation, the contents of rS are stored into the word in memory addressed by EA and the reservation is cleared.

If a reservation exists, but the memory address specified by the **stwcx.** instruction is not the same as that specified by the load and reserve instruction that established the reservation, the reservation is cleared, and it is undefined whether the contents of **r**S are stored into the word in memory addressed by EA.

If no reservation exists, the instruction completes without altering memory.

The CR0 field is set to reflect whether the store operation was performed as follows.

```
CR0[LT GT EQ S0] = 0b00 || store_performed ||
XER[SO]
```

EA must be a multiple of four. If it is not, either the system alignment exception handler is invoked or the results are boundedly undefined.

The granularity with which reservations are managed is *implementation-dependent*. Therefore, the memory to be accessed by the load and reserve and store conditional instructions should be allocated by a system library program.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

stwu

Assembler Syntax stwu rS,d(rA)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	37						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	d															

Definition Store Word with Update

Operation

EA ← (**r**A) + EXTS(d)
MEM(EA, 4) ← **r**S
rA ← EA

Description EA is the sum (**r**A) + d. The contents of **r**S are stored into the word in memory addressed by EA. EA is placed into **r**A. If **r**A = 0, the instruction form is invalid.

Other registers altered:

- None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

stwux

Assembler Syntax stwux rS,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						183									0

Definition Store Word with Update Indexed

Operation

EA ← (**r**A) + (**r**B)
MEM(EA, 4) ← **r**S
rA ← EA

Description EA is the sum (**r**A) + (**r**B). The contents of **r**S are stored into the word in memory addressed by EA. EA is placed into **r**A. If **r**A = 0, the instruction form is invalid.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

stwx

Assembler Syntax stwx rS,rA,rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						151									0

Definition Store Word Indexed

Operation

if **r**A = 0 then b ← 0
else b ← (**r**A)
EA ← b + (**r**B)
MEM(EA, 4) ← **r**S

Description EA is the sum (**r**A|0) + (**r**B). The contents of **r**S are is stored into the word in memory addressed by EA.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

subf

Assembler Syntax

subf	rD,rA,rB (OE = 0 Rc = 0)
subf.	rD,rA,rB (OE = 0 Rc = 1)
subfo	rD,rA,rB (OE = 1 Rc = 0)
subfo.	rD,rA,rB (OE = 1 Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					OE	40									RC

Definition Subtract From

Operation **r**D ← ¬ (**r**A) + (**r**B) + 1

Description The sum ¬ (**r**A) + (**r**B) + 1 is placed into **r**D. The **subf** instruction is preferred for subtraction because it sets few status bits.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
- XER:
  Affected: SO, OV (if OE = 1)

Simplified mnemonics:

sub	**r**D,**r**A,**r**B	equivalent to	**subf**	**r**D,**r**B,**r**A

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XO

subfc

Assembler Syntax

subfc	rD,rA,rB (OE = 0 Rc = 0)
subfc.	rD,rA,rB (OE = 0 Rc = 1)
subfco	rD,rA,rB (OE = 1 Rc = 0)
subfco.	rD,rA,rB (OE = 1 Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					OE	8									RC

Definition Subtract from Carrying

Operation **r**D ← ¬ (**r**A) + (**r**B) + 1

Description The sum ¬ (**r**A) + (**r**B) + 1 is placed into **r**D.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
  The CR0 field may not reflect the "true" (infinitely precise) result if overflow occurs (see XER below).
- XER:
  Affected: CA
  Affected: SO, OV (if OE = 1)

Simplified mnemonics:

subc	**r**D,**r**A,**r**B	equivalent to	**subfc**	**r**D,**r**B,**r**A

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XO

subfe

Assembler Syntax

subfe	rD,rA,rB (OE = 0 Rc = 0)
subfe.	rD,rA,rB (OE = 0 Rc = 1)
subfeo	rD,rA,rB (OE = 1 Rc = 0)
subfeo.	rD,rA,rB (OE = 1 Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					OE	136									RC

Definition Subtract from Extended

Operation **r**D ← ¬ (**r**A) + (**r**B) + XER[CA]

Description The sum ¬ (**r**A) + (**r**B) + XER[CA] is placed into **r**D.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
  The CR0 field may not reflect the "true" (infinitely precise) result if overflow occurs (see XER below).
- XER:
  Affected: CA
  Affected: SO, OV (if OE = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XO

subfic

Assembler Syntax subfic rD,rA,SIMM

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	08						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	SIMM															

Definition Subtract from Immediate Carrying

Operation **r**D ← ¬ (**r**A) + EXTS(SIMM) + 1

Description The sum ¬ (**r**A) + EXTS(SIMM) + 1 is placed into **r**D.

Other registers altered:

- XER:
  Affected: CA

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

subfme

Assembler Syntax

subfme	rD,rA	(OE = 0 Rc = 0)
subfme.	rD,rA	(OE = 0 Rc = 1)
subfmeo	rD,rA	(OE = 1 Rc = 0)
subfmeo.	rD,rA	(OE = 1 Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000					OE	232									RC

Definition

Subtract from Minus One Extended

Operation

rD ← ¬ (**r**A) + XER[CA] – 1

Description

The sum ¬ (**r**A) + XER[CA] + (32)1 is placed into **r**D.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO(if Rc = 1)
  The CR0 field may not reflect the "true" (infinitely precise) result if overflow occurs (see XER below).
- XER:
  Affected: CA
  Affected: SO, OV (if OE = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XO

subfze

Assembler Syntax

subfze	rD,rA	(OE = 0 Rc = 0)
subfze.	rD,rA	(OE = 0 Rc = 1)
subfzeo	rD,rA	(OE = 1 Rc = 0)
subfzeo.	rD,rA	(OE = 1 Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						D					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000					OE	200									RC

Definition Subtract from Zero Extended

Operation **r**D ← ¬ (**r**A) + XER[CA]

Description The sum ¬ (**r**A) + XER[CA] is placed into **r**D.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)
  The CR0 field may not reflect the "true" (infinitely precise) result if overflow occurs (see XER below).
- XER:
  Affected: CA
  Affected: SO, OV (if OE = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			XO

sync

Assembler Syntax sync

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						00000					00000				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000						598									0

Definition Synchronize

Description

The **sync** instruction provides an ordering function for the effects of all instructions executed by a given processor. Executing a **sync** instruction ensures that all instructions preceding the **sync** instruction appear to have completed before the **sync** instruction completes, and that no subsequent instructions are initiated by the processor until after the **sync** instruction completes. When the **sync** instruction completes, all external accesses caused by instructions preceding the **sync** instruction will have been performed with respect to all other mechanisms that access memory.

Multiprocessor implementations also send a **sync** address-only broadcast that is useful in some designs. For example, if a design has an external buffer that reorders loads and stores for better bus efficiency, the **sync** broadcast signals to that buffer that previous loads/stores must be completed before any following loads/stores. The **sync** instruction can be used to ensure that the results of all stores into a data structure, caused by store instructions executed in a "critical section" of a program, are seen by other processors before the data structure is seen as unlocked. The functions performed by the **sync** instruction will normally take a significant amount of time to complete, so indiscriminate use of this instruction may adversely affect performance. In addition, the time required to execute **sync** may vary from one execution to another. The **eieio** instruction may be more appropriate than **sync** for many cases.

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

tlbia

Assembler Syntax tlbia

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						00000					00000				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000						370									0

Definition Translation Lookaside Buffer Invalidate All

Operation All TLB entries ← invalid

Description The entire *translation lookaside buffer (TLB)* is invalidated (that is, all entries are removed). The TLB is invalidated regardless of the settings of MSR[IR] and MSR[DR]. The invalidation is done without reference to the SLB, segment table, or segment registers. This instruction does not cause the entries to be invalidated in other processors. This is a supervisor-level instructon.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
OEA	√	√	X

tlbie

Assembler Syntax tlbie rB

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						00000					00000				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B						30K6									0

Definition Translation Lookaside Buffer Invalidate Entry

Operation

VPS ← **r**B[4-19]
Identify TLB entries corresponding to VPS
Each such TLB entry ← invalid

Description

EA is the contents of **r**B. If the translation lookaside buffer (TLB) contains an entry corresponding to EA, that entry is made invalid (that is, removed from the TLB).

Multiprocessing implementations (for example, the 601, and 604) send a **tlbie** address-only broadcast over the address bus to tell other processors to invalidate the same TLB entry in their TLBs.

The TLB search is done regardless of the settings of MSR[IR] and MSR[DR]. The search is done based on a portion of the logical page number within a segment, without reference to the segment registers. All entries matching the search criteria are invalidated.

Block address translation for EA, if any, is ignored. This is a supervisor-level instruction.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
OEA	√	√	X

tlbsync

Assembler Syntax tlbsync

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						00000					00000				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	00000						566									0

Definition TLB Synchronize

Description If an implementation sends a broadcast for **tlbie** then it will also send a broadcast for **tlbsync**. Executing a **tlbsync** instruction ensures that all **tlbie** instructions previously executed by the processor executing the **tlbsync** instruction have completed on all other processors. The operation performed by this instruction is treated as a caching-inhibited and guarded data access with respect to the ordering done by **eieio**. This instruction is supervisor-level.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
OEA	√	√	X

tw

Assembler Syntax tw TO,rA,rB

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

Definition Trap Word

Operation

a ← EXTS(**r**A)
b ← EXTS(**r**B)
if (a < b) & TO[0] then TRAP
if (a > b) & TO[1] then TRAP
if (a = b) & TO[2] then TRAP
if (a <U b) & TO[3] then TRAP
if (a >U b) & TO[4] then TRAP

Description

The contents of **r**A are compared with the contents of **r**B. If any bit in the TO field is set and its corresponding condition is met by the result of the comparison, then the system trap handler is invoked.

Other registers altered:

- None

Simplified mnemonics:

tweq	**r**A,**r**B	equivalent to	**tw**	**4,r**A,**r**B
twlge	**r**A,**r**B	equivalent to	**tw**	**5,r**A,**r**B
trap		equivalent to	**tw**	**31,0,0**

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

twi

Assembler Syntax twi TO,rA,SIMM

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	03						TO					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	SIMM															

Definition Trap Word Immediate

Operation

a ← EXTS(**r**A)
if (a < EXTS(SIMM)) & TO[0] then TRAP
if (a > EXTS(SIMM)) & TO[1] then TRAP
if (a = EXTS(SIMM)) & TO[2] then TRAP
if (a <U EXTS(SIMM)) & TO[3] then TRAP
if (a >U EXTS(SIMM)) & TO[4] then TRAP

Description

The contents of **r**A are compared with the sign-extended value of the SIMM field. If any bit in the TO field is set and its corresponding condition is met by the result of the comparison, then the system trap handler is invoked.

Other registers altered:

- ❑ None

Simplified mnemonics:

twgti	**r**A,value	equivalent to	**twi**	**8,r**A,value
twllei	**r**A,value	equivalent to	**twi**	**6,r**A,value

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

xor

Assembler Syntax

xor	rA,rS,rB (Rc = 0)
xor.	rA,rS,rB (Rc = 1)

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	31						S					A				
BIT	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FIELD	B					316										RC

Definition XOR

Operation **r**A ← (**r**S) ⊕ (**r**B)

Description The contents of **r**S is XORed with the contents of **r**B and the result is placed into **r**A.

Other registers altered:

- Condition Register (CR0 field):
  Affected: LT, GT, EQ, SO (if Rc = 1)

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			X

xori

Assembler Syntax xori rA,rS,UIMM

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

Definition XOR Immediate

Operation **r**A ← (**r**S) ⊕ ((16)0 || UIMM)

Description The contents of **r**S are XORed with 0x0000 || UIMM and the result is placed into **r**A.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

xoris

Assembler Syntax xoris rA,rS,UIMM

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

Definition XOR Immediate Shifted

Operation **r**A ← (**r**S) ⊕ (UIMM || (16)0)

Description The contents of **r**S are XORed with UIMM || 0x0000 and the result is placed into **r**A.

Other registers altered:

- ❑ None

POWERPC ARCHITECTURE LEVEL	SUPERVISOR LEVEL	OPTIONAL	FORM
UISA			D

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

1ATX45182-0 Printed in USA 4/98 BANTA CO. MOTO #132 10,000 LITRISC