MPC855TUM/D
4/2002
REV 0.1

MPC855T User's Manual

Integrated Communications Microprocessor

freescale semiconductor

Home Page:
www.freescale.com
email:
support@freescale.com
USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
(800) 521-6274
480-768-2130
support@freescale.com
Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com
Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064, Japan
0120 191014
+81 2666 8080
support.japan@freescale.com
Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com
For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
(800) 441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

freescale™
semiconductor

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Chapter 11
Reset

Contents

Contents

Chapter 14
Clocks and Power Control

Contents

Chapter 15
Memory Controller

Contents

Contents

Chapter 16
PCMCIA Interface

Contents

Contents

Chapter 19
SDMA Channels and IDMA Emulation

Contents

Chapter 20
Serial Interface

Contents

Contents

Chapter 23
SCC HDLC Mode

Contents

Contents

Contents

Contents

Contents

Chapter 30
Serial Peripheral Interface (SPI)

Contents

Contents

Chapter 33
Parallel I/O Ports

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figure Number	Title	Page Number

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

About This Book

The primary objective of this manual is to help communications system designers build systems using the Motorola MPC855T and to help software designers provide operating systems and user-level applications to take fullest advantage of the MPC855T.

Although this book describes aspects regarding the PowerPC architecture that are critical for understanding the MPC8xx core, it does not contain a complete description of the architecture. Where additional information might help the reader, references are made to *Programming Environments Manual for the PowerPC Architecture*. Ordering information for this book are provided in the section “Related Documentation.”

The information in this book is subject to change without notice, as described in the disclaimers on the title page of this book. As with any technical documentation, it is the readers’ responsibility to be sure they are using the most recent version of the documentation. Contact your sales representative for more information.

Before Using This Manual

Before using this manual, determine whether it is the latest revision and if there are errata or addenda. To locate any published errata or updates for this document, refer to the world-wide web at http://www.motorola.com.

Audience

This manual is intended for software and hardware developers and application programmers who want to develop products for the MPC855T. It is assumed that the reader has a basic understanding of computer networking, OSI layers, and RISC architecture. In addition, it is assumed that the reader has a basic understanding of the communications protocols described here. Where it is considered useful, additional sources are provided that provide in-depth discussions of such topics.

Organization

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

- Part I, “Overview,” provides a high-level description of the MPC855T, describing general operation and listing basic features.

— Chapter 1, “MPC855T Overview, ” provides a high-level description of MPC855T functions and features. It roughly follows the structure of this book, summarizing the relevant features and providing references for the reader who needs additional information.

— Chapter 2, “Memory Map,” presents a table showing where MPC855T registers are mapped in memory. It includes cross references that indicate where each register is described in detail.

- Part II, “MPC8xx Microprocessor Module,” describes the MPC8xx core. These chapters provide details concerning the processor core as an implementation of the PowerPC architecture.
  — Chapter 3, “The MPC8xx Core,” provides an overview of the MPC855T core.
  — Chapter 4, “MPC8xx Core Register Set,” describes the hardware registers accessible to the MPC855T core. These include both architecturally-defined and MPC855T-specific registers.
  — Chapter 5, “MPC855T Instruction Set,” describes the PowerPC instructions implemented by the MPC855T. These instructions are organized by the level of architecture in which they are implemented—UISA, VEA, and OEA.
  — Chapter 6, “Exceptions,” describes the PowerPC exception model as it is implemented on the MPC855T.
  — Chapter 7, “Instruction and Data Caches,” describes the organization of the on-chip instruction and data caches, cache control, various cache operations, and the interaction between the caches, the load/store unit (LSU), the instruction sequencer, and the system interface unit (SIU).
  — Chapter 8, “Memory Management Unit” describes how the PowerPC MMU model is implemented on the MPC855T. Although the MPC855T MMU is based on the PowerPC MMU model, it differs greatly in many respects, which are described in this chapter.
  — Chapter 9, “Instruction Execution Timing,” describes the MPC855T instruction unit, and provides ways to make greatest advantage of its RISC architecture characteristics, such as pipelining and parallel execution. It includes a table of instruction latencies and lists dependencies and potential bottlenecks.
- Part III, “Configuration and Reset,” describes start-up behavior of the MPC855T.
  — Chapter 10, “System Interface Unit,” describes the SIU, which controls system start-up, initialization and operation, protection, as well as the external system bus.
  — Chapter 11, “Reset,” describes the behavior of the MPC855T at reset and start-up.
- Part IV, “Hardware Interface,” describes external signals, clocking, memory control, and power management of the MPC855T.

- Chapter 12, “External Signals,” provides a detailed description of the external signals that comprise the MPC855T external interface.
- Chapter 13, “External Bus Interface,” describes interactions among signals described in the previous chapter, including numerous examples and timing diagrams.
- Chapter 14, “Clocks and Power Control,” describes on-chip and external devices, including the phase-locked loop circuitry and frequency dividers that generate programmable clock timing for baud-rate generators, timers, and a variety of low-power mode options.
- Chapter 15, “Memory Controller,” describes the memory controller, which controlling a maximum of eight memory banks shared between a general-purpose chip-select machine (GPCM) and a pair of user-programmable machines (UPMs).
- Chapter 16, “PCMCIA Interface,” describes the PCMCIA host adapter module, which provides all control logic for a PCMCIA socket interface and requires only additional external analog power switching logic and buffering.

- Part V, “Communications Processor Module,” describes the configuration, clocking, and operation of the various communications protocols supported by the MPC855T.
  - Chapter 17, “Communications Processor Module and CPM Timers,” provides a brief overview of the MPC855T CPM and a detailed discussion of the clocking mechanisms supported.
  - Chapter 18, “Communications Processor,” describes the RISC communications processor (CP), which handles the low-level communications tasks, freeing the core for higher-level tasks.
  - Chapter 19, “SDMA Channels and IDMA Emulation,” describes the two physical serial DMA (SDMA) channels on the MPC855T with which the CP implements virtual SDMA channels.
  - Chapter 20, “Serial Interface,” describes the serial interface (SI) in which the physical interface to the SCC and SMCs is implemented.
  - Chapter 21, “Serial Communications Controller,” describes the serial communications controller (SCC1), which can be configured independently to implement different protocols for bridging functions, routers, and gateways, and to interface with a wide variety of standard WANs, LANs, and proprietary networks.
  - Chapter 22, “SCC UART Mode,” describes the MPC855T implementation of universal asynchronous receiver transmitter (UART) protocol, used for sending low-speed data between devices.
  - Chapter 23, “SCC HDLC Mode,” describes the MPC855T implementation of HDLC protocol.
  - Chapter 24, “SCC AppleTalk Mode,” describes the MPC855T implementation of AppleTalk, a set of protocols developed by Apple Computer, Inc. to provide a LAN service between Macintosh computers and printers.

- Chapter 25, “SCC Asynchronous HDLC Mode and IrDA,” describes the asynchronous HDLC and IrDA use of HDLC framing techniques with UART-type characters.
- Chapter 26, “SCC BISYNC Mode,” describes the MPC855T implementation of byte-oriented BISYNC protocol developed by IBM for use in networking products.
- Chapter 27, “SCC Ethernet Mode,” describes the MPC855T implementation of Ethernet protocol.
- Chapter 28, “SCC Transparent Mode,” describes the MPC855T implementation of transparent mode (also called totally transparent mode), which provides a clear channel on which the SCC can send or receive serial data without bit-level manipulation.
- Chapter 29, “Serial Management Controllers (SMCs),” describes two serial management controllers, full-duplex ports that can be configured independently to support one of three protocols—UART, transparent, or general-circuit interface (GCI).
- Chapter 30, “Serial Peripheral Interface (SPI),” describes the serial peripheral interface, which allows the MPC855T to exchange data between other MPC855T chips, the MC68360, the MC68302, the M68HC11 and M68HC05 microcontroller families, and peripheral devices such as EEPROMs, real-time clocks, A/D converters.
- Chapter 31, “I2C Controller,” describes the MPC855T implementation of the inter-integrated circuit (I²C®) controller, which allows data to be exchanged with other I²C devices, such as microcontrollers, EEPROMs, real-time clock devices, and A/D converters.
- Chapter 32, “Parallel Interface Port (PIP),” describes the parallel interface port which allows data to be sent to and from the MPC855T over 8 or 16 parallel data lines with two handshake control signals.
- Chapter 33, “Parallel I/O Ports,” describes the four general-purpose I/O ports—A, B, C, and D. Each signal in the I/O ports can be configured as a general-purpose I/O signal or as a signal dedicated to supporting communications devices, such as SMCs and the SCC.
- Chapter 34, “CPM Interrupt Controller,” describes how the CPM interrupt controller (CPIC) accepts and prioritizes the internal and external interrupt requests from the CPM blocks and passes them to the system interface unit (SIU). The CPIC also provides a vector during the core interrupt acknowledge cycle.

- Part VI, “Asynchronous Transfer Mode (ATM),” describes ATM implementation. It consists of the following chapters:
  - Chapter 35, “ATM Overview,” gives a high-level description of the MPC855T ATM implementation.

— Chapter 36, “Buffer Descriptors and Connection Tables,” describes the structure and configuration of the buffer descriptors (BDs) and the transmit and receive connection tables (TCTs and RCTs) used with ATM.

— Chapter 37, “ATM Parameter RAM,” describes how the parameter RAM is used to configure the SCC for serial ATM and the UTOPIA interface. The CP also uses parameter RAM to store operational and temporary values used during SAR activities.

— Chapter 38, “ATM Controller,” describes the address mapping mechanisms of the ATM controller to support connection tables for single- interfaces, and the commands provided to control ATM transmit and receive operations on a channel-by-channel basis.

— Chapter 39, “ATM Pace Control,” describes how the ATM pace control unit (APC) processes traffic parameters of each channel and defines the multiplex timing for all the channels.

— Chapter 40, “ATM Exceptions,” describes how the circular ATM interrupt queue operates with an event register (SCCE or IDSR1) to provide an interrupt model for ATM operations.

— Chapter 41, “Interface Configuration,” describes the programming of registers and parameters for ATM operations through both the UTOPIA and serial interfaces.

— Chapter 42, “UTOPIA Interface,” describes the classic SAR MPHY ATM operation, including the UTOPIA modes and the signals provided for UTOPIA support.

- Part VII, “Fast Ethernet Controller (FEC),” describes the MPC855Tsupport for 10/100 base-T Ethernet. It consists of the following chapter:

  — Chapter 43, “Fast Ethernet Controller (FEC),” describes the Fast Ethernet controller. It provides general descriptions of supported operations, full descriptions of the supporting registers, and initialization information.

- Part VIII, “System Debugging and Testing Support,” describes how to use the MPC855T facilities for debugging and system testing.

  — Chapter 44, “System Development and Debugging,” describes support provided for program flow tracking, internal watchpoint and breakpoint generation, and emulation systems control.

  — Chapter 45, “IEEE 1149.1 Test Access Port,” describes the dedicated user-accessible test access port (TAP), which is fully compatible with the *IEEE 1149.1 Standard Test Access Port and Boundary Scan Architecture*.

- Appendix A, “Byte Ordering,” discusses the MPC855T implementation of little- and big-endian byte mapping.

- Appendix B, “Serial Communications Performance,”provides guidelines for maximizing performance of MPC855T-based systems.
- Appendix C, “Register Quick Reference Guide,” contains a quick reference guide to the MPC855T- registers.
- Appendix D, “Instruction Set Listings,” contains tables of the PowerPC instructions supported by the MPC855T.
- Appendix E, “Serial ATM Scrambling, Reception, and SI Programming,” describes payload rescrambling and the MPC855Treceive procedure. It also provides a serial interface programming example.

This manual also includes a glossary and an index.

Suggested Reading

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

MPC8xx Documentation

Supporting documentation for the MPC855T can be accessed through the world-wide web at http://www.motorola.com. This documentation includes technical specifications, reference materials, and detailed applications notes.

Related Documentation

The documentation is organized in the following types of documents:

- *Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture* (MPEFPC32B/AD)—Describes resources defined by the PowerPC architecture.
- User’s manuals—These books provide details about individual implementations and are intended for use with the *Programming Environments Manual.*
- Addenda/errata to user’s manuals—Because some processors have follow-on parts an addendum is provided that describes the additional features and functionality changes. These addenda are intended for use with the corresponding user’s manuals.
- Hardware specifications—Hardware specifications provide specific data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics, as well as other design considerations. Separate hardware specifications are provided for each part described in this book.
- Technical summaries—Each device has a technical summary that provides an overview of its features. This document is roughly the equivalent to the overview (Chapter 1) of an implementation’s user’s manual.

- *The Programmer's Reference Guide for the PowerPC Architecture*: MPCPRG/D—This concise reference includes the register summary, memory control model, exception vectors, and the PowerPC instruction set.
- *The Programmer's Pocket Reference Guide for the PowerPC Architecture*: MPCPRGREF/D—This foldout card provides an overview of PowerPC registers, instructions, and exceptions for 32-bit implementations.
- Application notes—These short documents address specific design issues useful to programmers and engineers working with Motorola processors.

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

Conventions

This document uses the following notational conventions:

Bold	Bold entries in figures and tables showing registers and parameter RAM should be initialized by the user.
mnemonics	Instruction mnemonics are shown in lowercase bold.
italics	Italics indicate variable command parameters, for example, **bcctr***x*. Book titles in text are set in italics.
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote binary number
rA, **r**B	Instruction syntax used to identify a source GPR
rD	Instruction syntax used to identify a destination GPR
REG[FIELD]	Abbreviations or acronyms for registers or buffer descriptors are shown in uppercase text. Specific bits, fields, or numerical ranges appear in brackets. For example, MSR[LE] refers to the little-endian mode enable bit in the machine state register.
x	In certain contexts, such as in a signal encoding or a bit field, indicates a don't care.
n	Used to express an undefined numerical value
¬	NOT logical operator
&	AND logical operator
|	OR logical operator

Acronyms and Abbreviations

Table i contains acronyms and abbreviations used in this document. Note that the meanings for some acronyms (such as SDR1 and DSISR) are historical, and the words for which an acronym stands may not be intuitively obvious.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
A/D	Analog-to-digital
ALU	Arithmetic logic unit
ATM	Asynchronous transfer mode
BD	Buffer descriptor
BIST	Built-in self test
BPU	Branch processing unit

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
BRI	Basic rate interface.
BUID	Bus unit ID
CAM	Content-addressable memory
CEPT	Conference des administrations Europeanes des Postes et Telecommunications (European Conference of Postal and Telecommunications Administrations).
CP	Communications processor
CPM	Communications processor module
CR	Condition register
CRC	Cyclic redundancy check
CTR	Count register
DABR	Data address breakpoint register
DAR	Data address register
DEC	Decrementer register
DMA	Direct memory access
DPLL	Digital phase-locked loop
DRAM	Dynamic random access memory
DSISR	Register used for determining the source of a DSI exception
DTLB	Data translation lookaside buffer
EA	Effective address
EEST	Enhanced Ethernet serial transceiver
EPROM	Erasable programmable read-only memory
FPR	Floating-point register
FPSCR	Floating-point status and control register
FPU	Floating-point unit
GCI	General circuit interface
GPCM	General-purpose chip-select machine
GPR	General-purpose register
GUI	Graphical user interface
HDLC	High-level data link control
I^2C	Inter-integrated circuit
IDL	Inter-chip digital link
IEEE	Institute of Electrical and Electronics Engineers

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
IrDA	Infrared Data Association
ISDN	Integrated services digital network
ITLB	Instruction translation lookaside buffer
IU	Integer unit
JTAG	Joint test action group
LIFO	Last-in-first-out
LR	Link register
LRU	Least recently used
LSB	Least-significant byte
lsb	Least-significant bit
LSU	Load/store unit
MAC	Multiply accumulate
MESI	Modified/exclusive/shared/invalid—cache coherency protocol
MMU	Memory management unit
MSB	Most-significant byte
msb	Most-significant bit
MSR	Machine state register
NaN	Not a number
NIA	Next instruction address
NMSI	Nonmultiplexed serial interface
No-op	No operation
OEA	Operating environment architecture
OSI	Open systems interconnection
PCI	Peripheral component interconnect
PCMCIA	Personal Computer Memory Card International Association
PIR	Processor identification register
PRI	Primary rate interface
PVR	Processor version register
RISC	Reduced instruction set computing
RTOS	Real-time operating system
RWITM	Read with intent to modify

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
Rx	Receive
SCC	Serial communications controller
SCP	Serial control port
SDLC	Synchronous Data Link Control
SDMA	Serial DMA
SI	Serial interface
SIMM	Signed immediate value
SIU	System interface unit
SMC	Serial management controller
SNA	Systems network architecture
SPI	Serial peripheral interface
SPR	Special-purpose register
SPRG*n*	Registers available for general purposes
SRAM	Static random access memory
SRR0	Machine status save/restore register 0
SRR1	Machine status save/restore register 1
TAP	Test access port
TB	Time base register
TDM	Time-division multiplexed
TLB	Translation lookaside buffer
TSA	Time-slot assigner
Tx	Transmit
UART	Universal asynchronous receiver/transmitter
UIMM	Unsigned immediate value
UISA	User instruction set architecture
UPM	User-programmable machine
USART	Universal synchronous/asynchronous receiver/transmitter
VA	Virtual address
VEA	Virtual environment architecture
XER	Register used primarily for indicating conditions such as carries and overflows for integer operations

PowerPC Architecture Terminology Conventions

Table ii lists certain terms used in this manual that differ from the architecture terminology conventions.

Table ii. Terminology Conventions

The Architecture Specification	This Manual
Data storage interrupt (DSI)	DSI exception
Extended mnemonics	Simplified mnemonics
Instruction storage interrupt (ISI)	ISI exception
Interrupt	Exception
Privileged mode (or privileged state)	Supervisor-level privilege
Problem mode (or problem state)	User-level privilege
Real address	Physical address
Relocation	Translation
Storage (locations)	Memory
Storage (the act of)	Access

Table iii describes instruction field notation conventions used in this manual.

Table iii. Instruction Field Conventions

The Architecture Specification	Equivalent to:
BA, BB, BT	**crb**A, **crb**B, **crb**D (respectively)
BF, BFA	**crf**D, **crf**S (respectively)
D	d
DS	ds
FLM	FM
FXM	CRM
RA, RB, RT, RS	**r**A, **r**B, **r**D, **r**S (respectively)
SI	SIMM
U	IMM
UI	UIMM
/, //, ///	0...0 (shaded)

Part I
Overview

Intended Audience

Part I is intended for anyone who requires a high-level understanding of the MPC855T family of PowerQUICC devices.

Contents

Part I provides an overview of the features and functions of the MPC855T. It includes the following chapters:

- Chapter 1, “MPC855T Overview, ” provides a high-level description of MPC855T functions and features. It roughly follows the structure of this book, summarizing the relevant features and providing references for the reader who needs additional information.
- Chapter 2, “Memory Map,” presents a table showing where MPC855T registers are mapped in memory. It includes cross references that indicate where each register is described in detail.

Conventions

Part I uses the following notational conventions:

mnemonics	Instruction mnemonics are shown in lowercase bold.
italics	Italics indicate variable command parameters, for example, **bcctr***x*. Book titles in text are set in italics.
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote binary number
rA, **r**B	Instruction syntax used to identify a source GPR
rD	Instruction syntax used to identify a destination GPR

REG[FIELD]	Abbreviations or acronyms for registers or buffer descriptors are shown in uppercase text. Specific bits, fields, or numerical ranges appear in brackets. For example, MSR[LE] refers to the little-endian mode enable bit in the machine state register.
x	In certain contexts, such as in a signal encoding or a bit field, indicates a don’t care.
n	Indicates an undefined numerical value

Acronyms and Abbreviations

Table i contains acronyms and abbreviations that are used in this document.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
BD	Buffer descriptor
BPU	Branch processing unit
CP	Communications processor
CPM	Communications processor module
DMA	Direct memory access
DPLL	Digital phase-locked loop
DRAM	Dynamic random access memory
DTLB	Data translation lookaside buffer
EA	Effective address
GPCM	General-purpose chip-select machine
GPR	General-purpose register
HDLC	High-level data link control
I^2C	Inter-integrated circuit
IEEE	Institute of Electrical and Electronics Engineers
IrDA	Infrared Data Association
ISDN	Integrated services digital network
ITLB	Instruction translation lookaside buffer
IU	Integer unit
JTAG	Joint Test Action Group
LRU	Least recently used (cache replacement algorithm)
LSU	Load/store unit
MMU	Memory management unit
MSR	Machine state register
NMSI	Nonmultiplexed serial interface

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
OEA	Operating environment architecture
OSI	Open systems interconnection
PCI	Peripheral component interconnect
PCMCIA	Personal Computer Memory Card International Association
RISC	Reduced instruction set computing
RTOS	Real-time operating system
Rx	Receive
SCC	Serial communications controller
SDLC	Synchronous data link control
SDMA	Serial DMA
SI	Serial interface
SIU	System interface unit
SMC	Serial management controller
SPI	Serial peripheral interface
SPR	Special-purpose register
SRAM	Static random access memory
TB	Time base register
TDM	Time-division multiplexed
TLB	Translation lookaside buffer
TSA	Time-slot assigner
Tx	Transmit
UART	Universal asynchronous receiver/transmitter
UISA	User instruction set architecture
UPM	User-programmable machine
VEA	Virtual environment architecture

Chapter 1
MPC855T Overview

The MPC855T is a versatile one-chip integrated microprocessor and peripheral combination that can be used in a variety of controller applications. It particularly excels in both communications and networking systems. Unless otherwise specified, the PowerQUICC unit is referred to as the MPC855T in this manual.

The MPC855T is a PowerPC architecture-based derivative of Motorola's MPC860 Quad Integrated Communications Controller (PowerQUICC™). The CPU on the MPC855T is the MPC8xx core, a 32-bit microprocessor which implements the PowerPC architecture, incorporating memory management units (MMUs) and instruction and data caches.

The purpose of this manual is to describe the operation of MPC855T functionality with concentration on the I/O functions. Additional information can be found in *Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture.*

1.1 Features

The following list summarizes the key MPC855T features:

- Embedded MPC8xx core
- Single-issue, 32-bit core (compatible with the PowerPC architecture definition) with 32, 32-bit general-purpose registers (GPRs)
  - — The core performs branch prediction with conditional prefetch, without conditional execution
  - — 4-Kbyte data cache and 4- Kbyte instruction cache.
    - – 4-Kbyte instruction cache is two-way, set-associative with 128 sets.
    - – 4-Kbyte data cache is two-way, set-associative with 128 sets.
    - – Cache coherency for both instruction and data caches is maintained on 128-bit (4-word) cache blocks.
    - – Caches are physically addressed, implement a least recently used (LRU) replacement algorithm, and are lockable on a cache block basis.
  - — MMUs with 32-entry TLB, fully associative instruction and data TLBs
  - — MMUs support multiple page sizes of 4, 16, and 512 Kbytes, and 8 Mbytes; 16 virtual address spaces and 16 protection groups

  - Advanced on-chip-emulation debug mode
- The MPC855T provides the same ATM functionality as that of the MPC860SAR
- Up to 32-bit data bus (dynamic bus sizing for 8, 16, and 32 bits)
- 32 address lines
- Memory controller (eight banks)
  - Contains complete dynamic RAM (DRAM) controller
  - Each bank can be a chip select or $\overline{\text{RAS}}$ to support a DRAM bank
  - Up to 30 wait states programmable per memory bank
  - Glueless interface to DRAM, SIMMS, SRAM, EPROMs, flash EPROMs, and other memory devices.
  - DRAM controller programmable to support most size and speed memory interfaces
  - Four $\overline{\text{CAS}}$ lines, four $\overline{\text{WE}}$ lines, one $\overline{\text{OE}}$ line
  - Boot chip-select available at reset (options for 8-, 16-, or 32-bit memory)
  - Variable block sizes (32 Kbyte–256 Mbyte)
  - Selectable write protection
  - On-chip bus arbitration logic
- General-purpose timers
  - Four 16-bit timers or two 32-bit timers
  - Gate mode can enable/disable counting
  - Interrupt can be masked on reference match and event capture
- Fast Ethernet controller (FEC)
- System integration unit (SIU)
  - Bus monitor
  - Software watchdog
  - Periodic interrupt timer (PIT)
  - Low-power stop mode
  - Clock synthesizer
  - Decrementer and time base
  - Real-time clock (RTC)
  - Reset controller
  - IEEE 1149.1 test access port (JTAG)
- Interrupts
  - Seven external interrupt request (IRQ) lines
  - 12 port pins with interrupt capability

  - 20 internal interrupt sources
  - Programmable highest priority request
- Communications processor module (CPM)
  - RISC controller
  - Communication-specific commands (for example, GRACEFUL STOP TRANSMIT, ENTER HUNT MODE, and RESTART TRANSMIT)
  - Supports continuous mode transmission and reception on all serial channels
  - 8-Kbytes of dual-port RAM
  - 16 serial DMA (SDMA) channels
- Three parallel I/O registers with open-drain capability
- Four baud rate generators
  - Independent (can be connected to any SCC or SMC)
  - Allow changes during operation
  - Autobaud support option
- One SCC (serial communication controller)
  - Serial ATM capability
  - Ethernet/IEEE 802.3 supporting full 10-Mbps operation
  - HDLC/SDLC
  - HDLC bus (implements an HDLC-based local area network (LAN))
  - Asynchronous HDLC to support PPP (point-to-point protocol)
  - AppleTalk
  - Universal asynchronous receiver transmitter (UART)
  - Synchronous UART
  - Serial infrared (IrDA)
  - Binary synchronous communication (BISYNC)
  - Totally transparent (bit streams)
  - Totally transparent (frame based with optional cyclic redundancy check (CRC))
- Two SMCs (serial management channels)
  - UART
  - Transparent
  - General circuit interface (GCI) controller
  - Can be connected to the time-division multiplexed (TDM) channel
- One SPI (serial peripheral interface)
  - Supports master and slave modes
  - Supports multimaster operation on the same bus

- One I^2C (inter-integrated circuit) port
  - — Supports master and slave modes
  - — Multiple-master environment support
- Time-slot assigner (TSA)
  - — Allows SCC and SMCs to run in multiplexed and/or non-multiplexed operation
  - — Supports T1, CEPT, PCM highway, user defined
  - — 1- or 8-bit resolution
  - — Allows independent transmit and receive routing, frame synchronization, clocking
  - — Allows dynamic changes
  - — Can be internally connected to three serial channels (one SCC and two SMCs)
- Parallel interface port (PIP)
  - — Centronics interface support
  - — Supports fast connection between compatible ports on MPC855T or MC68360
- PCMCIA interface
  - — Master (socket) interface, release 2.1 compliant
  - — Supports two independent PCMCIA sockets
  - — 8 memory or I/O windows supported
- Low power support
  - — Full on—All units fully powered
  - — Doze—Core functional units disabled except time base decrementer, PLL, memory controller, RTC, and CPM in low-power standby
  - — Sleep—All units disabled except RTC, PIT, time base, and decrementer with PLL active for fast wake up
  - — Deep sleep—All units disabled including PLL except RTC, PIT, time base, and decrementer.
  - — Power down mode— All units powered down except RTC, PIT, time base and decrementer
- Debug interface
  - — Eight comparators: four operate on instruction address, two operate on data address, and two operate on data
  - — Supports conditions: = ≠ < >
  - — Each watchpoint can generate a break point internally
- 3.3 V operation
- 357-pin ball grid array (BGA) package

The MPC855T is comprised of three modules that each use the 32-bit internal bus: the MPC8xx core, the system integration unit (SIU), and the communication processor module (CPM). The MPC855T block diagram is shown in Figure 1-1.

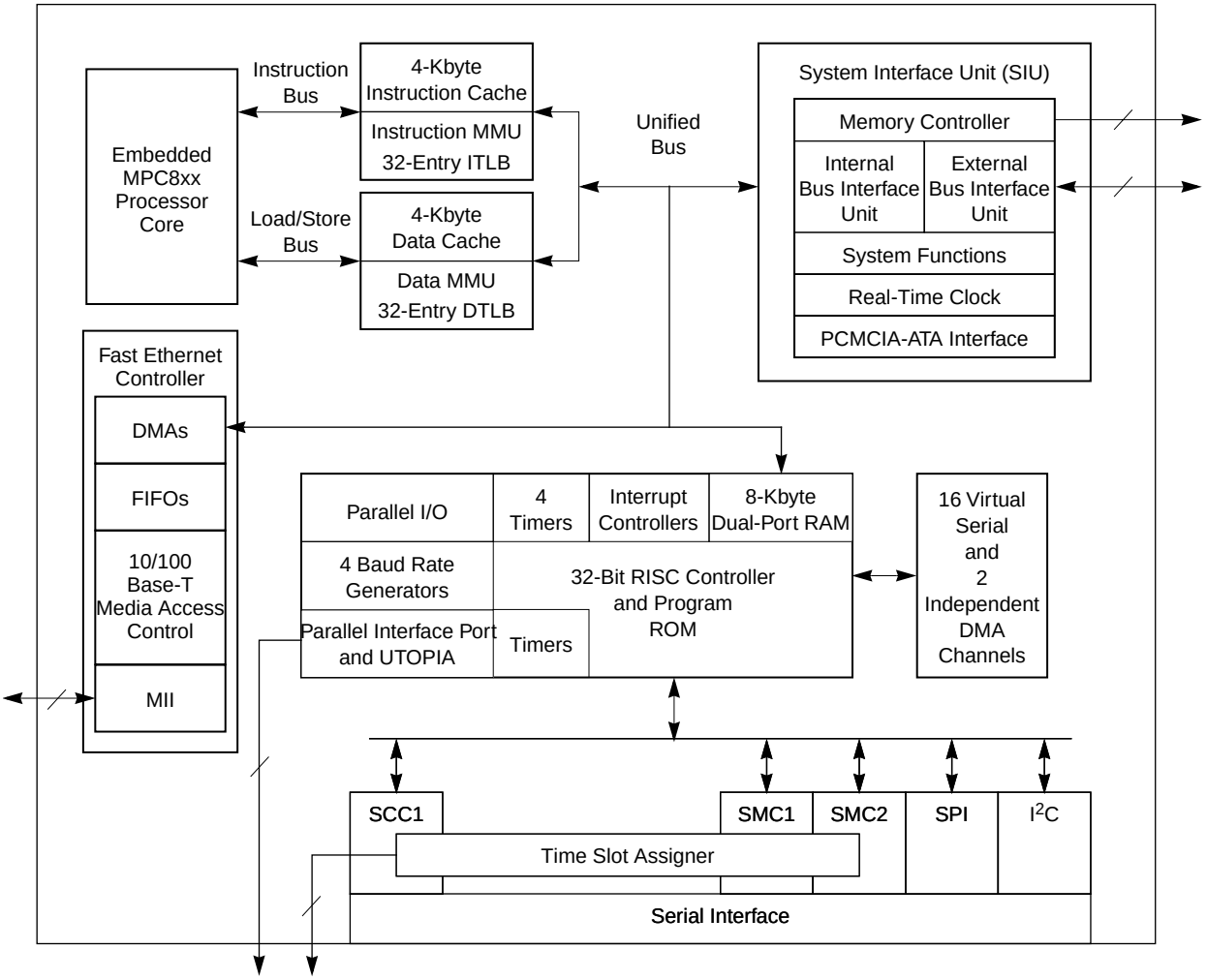


Figure 1-1. MPC855T Block Diagram

1.2 Embedded MPC8xx Core

The MPC855T integrates an embedded MPC8xx core with high-performance, low-power peripherals to extend the Motorola Data Communications family of embedded processors farther into high-end communications and networking products.

The core is compliant with the UISA (user instruction set architecture) portion of the PowerPC architecture. It has an integer unit (IU) and a load/store unit (LSU) that execute all integer and load/store operations in hardware. The core supports integer operations on a 32-bit internal data path and 32-bit arithmetic hardware. The core interface to the internal and external buses is 32 bits.

The IU uses 32, 32-bit GPRs for source and target operands. Typically, it can execute one integer instruction each clock cycle. Each element in the integer block is clocked only when valid data is in the data queue and is ready for operation. This holds power consumption of the device to the absolute minimum.

The core is integrated with MMUs as well as instruction and data caches. Each MMU provides a 32-entry, fully associative instruction and data TLB, with multiple page sizes of 4, 16, 512, and 256 Kbytes and 8 Mbytes. It supports 16 virtual address spaces with 8 protection groups. Three special scratch registers support software table search and update operations.

The instruction cache is four-way, set associative with physical addressing. It allows single-cycle access on hits with no added latency for misses. It has four words per block, supporting a four-beat burst line fill using an LRU (least recently used) replacement algorithm. The cache can be locked on a per cache block basis for application-critical routines.

The data cache is two-way, set associative with physical addressing. It allows single-cycle accesses on hits with one added clock latency for misses. It has four words per cache block, supporting burst line fill using LRU replacement. The cache can be locked on a per block basis for application critical routines. The data cache can be programmed through the MMU to support copy-back or write-through. Cache-inhibit mode can be programmed per MMU page.

The debug interface provides superior debug capabilities without degrading operation speed. This interface supports six watchpoint pins that are used to detect software events. Four of its eight internal comparators operate on the effective address on the address bus, two operate on the effective address on the data address bus, and two operate on the data bus. The core can make =, ≠, <, and > comparisons to generate watchpoints. Each watchpoint can then generate a break point that can be configured to trigger in a programmable number of events.

1.3 System Interface Unit (SIU)

The SIU on the MPC855T integrates general-purpose features useful in almost any 32-bit processor system. Dynamic bus sizing allows 8-, 16-, and 32-bit peripherals and memory to exist in the 32-bit system bus mode.

The SIU also provides power management functions, reset control, decrementer, timebase and the real-time clock.

The memory controller supports up to eight memory banks with glueless interfaces to DRAM, SRAM, SSRAM, EPROM, Flash EPROM, SDRAM, EDO, and other peripherals with 2-clock-cycle access to external SRAM and bursting support. It provides variable block sizes from 32 Kbytes to 256 Mbytes. The memory controller provides 0–30 wait states for each memory bank and can use address type matching to qualify each memory bank access. It provides four byte-enable signals, an output-enable signal. and a boot chip select available at reset.

The DRAM interface supports port sizes of 8, 16, and 32 bits. Memory banks can be defined in depths of 256 or 512 Kbytes or 1, 2, 4, 8, 16, 32, or 64 Mbytes for all port sizes.

The memory depth can be 64 and 128 Kbytes for 8-bit memory or 128 and 256 Mbytes for 32-bit memory. The DRAM controller supports page-mode access for successive transfers within bursts. The MPC855T supports a glueless interface to one bank of DRAM while external buffers are required for additional memory banks. The refresh unit provides $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$, a programmable refresh timer, refresh active during external reset, disable refresh mode, and stacking up to 7 refresh cycles. The DRAM interface uses a programmable state machine to support almost any memory interface.

1.4 PCMCIA Controller

The PCMCIA interface is a master (socket) controller and is compliant with release 2.1. The interface supports up to two independent PCMCIA sockets requiring only external transceivers/buffers. The interface provides eight memory or I/O windows where each window can be allocated to a particular socket. If only one PCMCIA port is used, the unused port may be used as general-purpose input with interrupt capability.

1.5 Power Management

The MPC855T supports a wide range of power management features including full on, doze, sleep, deep sleep, and low power stop.

- Full on mode leaves the MPC855T processor fully powered with all internal units operating at the full processor speed. A gear mode is determined by a clock divider, allowing the operating system to reduce the processor's operational frequency.
- Doze mode disables core functional units other than the time base decrementer, PLL, memory controller, RTC, and places the CPM in low-power standby mode.
- Sleep mode disables everything except the RTC and PIT, leaving the PLL active for quick wake-up.
- Deep sleep mode disables the PLL for lower power but slower wake-up.
- Low-power stop disables all logic in the processor except the minimum logic required to restart the device, providing the lowest power consumption but requiring the longest wake-up time.

1.6 Communications Processor Module (CPM)

The MPC855T is the next generation of the MPC8xx PowerQUICC family of devices. Like its predecessor it implements a dual-processor architecture, which provides both a high-performance, general-purpose processor for application programming use as well as a special-purpose communication processor (CPM) uniquely designed for communications applications.

The CPM contains features that, like its predecessor, allow the MPC855T to excel in communications and networking products. These features are grouped as follows:

- Communications processor (CP)
- Sixteen independent DMA (SDMA) controllers
- Four general-purpose timers

The CP provides the communication features of the MPC855T. Included are a RISC processor, one serial communication controller (SCC1), two serial management controllers (SMCs), a serial peripheral interface (SPI), an I^2C interface, 8 Kbytes of dual-port RAM, an interrupt controller, a time-slot assigner, three parallel ports, a parallel interface port, four independent baud rate generators, and sixteen serial DMA channels to support the SCC, SMCs, SPI, and I^2C.

The SDMAs provide two channels of general-purpose DMA capability for each communications channel. They offer high-speed transfers, 32-bit data movement, buffer chaining, and independent request and acknowledge logic.

The four general-purpose timers on the CPM are identical to the timers found on the MPC8xx, which support the internal cascading of two timers to form a 32-bit timer.

1.7 ATM Capabilities

The MPC855T can be used as an adaptable ATM controller suited for a variety of applications, including the following:

- ATM line card controllers
- ATM-to-WAN interworking, including frame relay, T1/E1 circuit emulation service (CES), and xDSL applications
- Residential broadband network interface units (ATM-to-Ethernet)
- Set-top controllers
- ATM25 applications
- Bridging and routing applications

Chapter 2
Memory Map

Each memory resource in the MPC855T is mapped within a contiguous block of 16 Kbyte memory. The location of this block within the global 4-Gbyte physical memory space can be mapped on 64-Kbyte resolution through an implementation-specific special-purpose register (SPR) called the internal memory map register (IMMR). See Section 10.4.1, “Internal Memory Map Register (IMMR).” Table 2-1 defines the internal memory map.

Table 2-1. MPC855T Internal Memory Map

Offset	Name	Size	Section/Page
General System Interface Unit			
000	SIUMCR—SIU module configuration register	32 bits	10.4.2/-5
004	SYPCR—System protection control register	32 bits	10.4.3/-7
008–00D	Reserved	6 bytes	—
00E	SWSR—Software service register	16 bits	10.7.1/-21
010	SIPEND—SIU interrupt pending register	32 bits	10.5.4.1/-15
014	SIMASK—SIU interrupt mask register	32 bits	10.5.4.2/-16
018	SIEL—SIU interrupt edge/level register	32 bits	10.5.4.3/-17
01C	SIVEC—SIU interrupt vector register	32 bits	10.5.4.4/-18
020	TESR—Transfer error status register	32 bits	10.4.4/-8
024–02F	Reserved	12 bytes	—
030	SDCR—SDMA configuration register	32 bits	19.2.1/-3
034–07F	Reserved	76 bytes	—
PCMCIA			
080	PBR0—PCMCIA interface base register 0	32 bits	16.4.5/-14
084	POR0—PCMCIA interface option register 0	32 bits	16.4.6/-14
088	PBR1—PCMCIA interface base register 1	32 bits	16.4.5/-14
08C	POR1—PCMCIA interface option register 1	32 bits	16.4.6/-14
090	PBR2—PCMCIA interface base register 2	32 bits	16.4.5/-14
094	POR2—PCMCIA interface option register 2	32 bits	16.4.6/-14

Table 2-1. MPC855T Internal Memory Map (continued)

Offset	Name	Size	Section/Page
098	PBR3—PCMCIA interface base register 3	32 bits	16.4.5/-14
09C	POR3—PCMCIA interface option register 3	32 bits	16.4.6/-14
0A0	PBR4—PCMCIA interface base register 4	32 bits	16.4.5/-14
0A4	POR4—PCMCIA interface option register 4	32 bits	16.4.6/-14
0A8	PBR5—PCMCIA interface base register 5	32 bits	16.4.5/-14
0AC	POR5—PCMCIA interface option register 5	32 bits	16.4.6/-14
0B0	PBR6—PCMCIA interface base register 6	32 bits	16.4.5/-14
0B4	POR6—PCMCIA interface option register 6	32 bits	16.4.6/-14
0B8	PBR7—PCMCIA interface base register 7	32 bits	16.4.5/-14
0BC	POR7—PCMCIA interface option register 7	32 bits	16.4.6/-14
0C0–0DF	Reserved	32 bytes	—
0E0	PGCRA—PCMCIA interface general control register A	32 bits	16.4.4/-13
0E4	PGCRB—PCMCIA interface general control register B	32 bits	16.4.4/-13
0E8	PSCR—PCMCIA interface status changed register	32 bits	16.4.2/-10
0EC–0EF	Reserved	4 bytes	—
0F0	PIPR—PCMCIA interface input pins register	32 bits	16.4.1/-9
0F4–0F7	Reserved	4 bytes	—
0F8	PER—PCMCIA interface enable register	32 bits	16.4.3/-11
0FC–0FF	Reserved	4 bytes	—
Memory Controller			
100	BR0—Base register bank 0	32 bits	15.4.1/-9
104	OR0—Option register bank 0	32 bits	15.4.2/-10
108	BR1—Base register bank 1	32 bits	15.4.1/-9
10C	OR1—Option register bank 1	32 bits	15.4.2/-10
110	BR2—Base register bank 2	32 bits	15.4.1/-9
114	OR2—Option register bank 2	32 bits	15.4.2/-10
118	BR3—Base register bank 3	32 bits	15.4.1/-9
11C	OR3—Option register bank 3	32 bits	15.4.2/-10
120	BR4—Base register bank 4	32 bits	15.4.1/-9
124	OR4—Option register bank 4	32 bits	15.4.2/-10
128	BR5—Base register bank 5	32 bits	15.4.1/-9

Table 2-1. MPC855T Internal Memory Map (continued)

Offset	Name	Size	Section/Page
12C	OR5—Option register bank 5	32 bits	15.4.2/-10
130	BR6—Base register bank 6	32 bits	15.4.1/-9
134	OR6—Option register bank 6	32 bits	15.4.2/-10
138	BR7—Base register bank 7	32 bits	15.4.1/-9
13C	OR7—Option register bank 7	32 bits	15.4.2/-10
140–163	Reserved	36 bytes	—
164	MAR—Memory address register	32 bits	15.4.7/-17
168	MCR—Memory command register	32 bits	15.4.5/-15
16C–16F	Reserved	4 bytes	—
170	MAMR—Machine A mode register	32 bits	15.4.4/-13
174	MBMR—Machine B mode register	32 bits	15.4.4/-13
178	MSTAT—Memory status register	16 bits	15.4.3/-13
17A	MPTPR—Memory periodic timer prescaler	16 bits	15.4.8/-18
17C	MDR—Memory data register	32 bits	15.4.6/-17
180–1FF	Reserved	128 bytes	—
System Integration Timers			
200	TBSCR—Timebase status and control register	16 bits	10.9.3/-25
202–203	Reserved	2 bytes	—
204	TBREFA—Timebase reference register A	32 bits	10.9.2/-24
208	TBREFB—Timebase reference register B	32 bits	
20C–21F	Reserved	20 bytes	—
220	RTCSC—Real-time clock status and control register	16 bits	10.10.1/-27
222–223	Reserved	2 bytes	—
224	RTC—Real-time clock register	32 bits	10.10.2/-28
228	RTSEC—Real-time alarm seconds	32 bits	10.10.4/-29
22C	RTCAL—Real-time alarm register	32 bits	10.10.3/-28
230–23F	Reserved	16 bytes	—
240	PISCR—Periodic interrupt status and control register	16 bits	10.11.1/-31
242–243	Reserved	2 bytes	—
244	PITC—Periodic interrupt count register	32 bits	10.11.2/-32
248	PITR—Periodic interrupt timer register	32 bits	10.11.3/-32
24C–27F	Reserved	52 bytes	—

Table 2-1. MPC855T Internal Memory Map (continued)

Offset	Name	Size	Section/Page
Clocks and Reset			
280	SCCR—System clock control register	32 bits	14.6.1/-29
284	PLPRCR—PLL, low-power, and reset control register	32 bits	14.6.2/-31
288	RSR—Reset status register	32 bits	11.2/-5
28C–2FF	Reserved	116 bytes	—
System Integration Timers Keys			
300	TBSCRK—Timebase status and control register key	32 bits	10.4.5/-9
304	TBREFAK—Timebase reference register A key	32 bits	10.4.5/-9
308	TBREFBK—Timebase reference register B key	32 bits	10.4.5/-9
30C	TBK—Timebase/decrementer register key	32 bits	10.4.5/-9
310–31F	Reserved	16 bytes	—
320	RTCSCK—Real-time clock status and control register key	32 bits	10.4.5/-9
324	RTCK—Real-time clock register key	32 bits	10.4.5/-9
328	RTSECK—Real-time alarm seconds key	32 bits	10.4.5/-9
32C	RTCALK—Real-time alarm register key	32 bits	10.4.5/-9
330–33F	Reserved	16 bytes	—
340	PISCRK—Periodic interrupt status and control register key	32 bits	10.4.5/-9
344	PITCK—Periodic interrupt count register key	32 bits	10.4.5/-9
348–37F	Reserved	56 bytes	—
Clocks and Reset Keys			
380	SCCRK—System clock control key	32 bits	10.4.5/-9
384	PLPRCRK—PLL, low power and reset control register key	32 bits	10.4.5/-9
388	RSRK—Reset status register key	32 bits	10.4.5/-9
38C–85F	Reserved	1236 bytes	—
I^2C Controller			
860	I2MOD—I^2C mode register	8 bits	31.4.1/-6
861–863	Reserved	3 bytes	—
864	I2ADD—I^2C address register	8 bits	31.4.2/-7
865–867	Reserved	3 bytes	—

Table 2-1. MPC855T Internal Memory Map (continued)

Offset	Name	Size	Section/Page
868	I2BRG—I^2C BRG register	8 bits	31.4.3/-8
869–86B	Reserved	3 bytes	—
86C	I2COM—I^2C command register	8 bits	31.4.5/-9
86D–86F	Reserved	3 bytes	—
870	I2CER—I^2C event register	8 bits	31.4.4/-8
871–873	Reserved	3 bytes	—
874	I2CMR—I^2C mask register	8 bits	31.4.4/-8
875–8FF	Reserved	139 bytes	—
DMA			
900–903	Reserved	4 bytes	—
904	SDAR—SDMA address register	32 bits	19.2.4/-5
908	SDSR—SDMA status register	8 bits	19.2.2/-4
909–90B	Reserved	3 bytes	—
90C	SDMR—SDMA mask register	8 bits	19.2.3/-5
90D–90F	Reserved	3 bytes	—
910	IDSR1—IDMA1 status register	8 bits	19.3.9.2/-21
911–913	Reserved	3 bytes	—
914	IDMR1—IDMA1 mask register	8 bits	19.3.9.3/-21
915–917	Reserved	3 bytes	—
918	IDSR2—IDMA2 status register	8 bits	19.3.9.2/-21
919–91B	Reserved	3 bytes	—
91C	IDMR2—IDMA2 mask register	8 bits	19.3.9.3/-21
91D–92F	Reserved	19 bytes	—
Communications Processor Module Interrupt Control			
930	CIVR—CPM interrupt vector register	16 bits	34.5.5/-9
932–93F	Reserved	14 bytes	—
940	CICR—CPM interrupt configuration register	32 bits	34.5.1/-7
944	CIPR—CPM interrupt pending register	32 bits	34.5.2/-7
948	CIMR—CPM interrupt mask register	32 bits	34.5.3/-8
94C	CISR—CPM in-service register	32 bits	34.5.4/-8
Input/Output Port			
950	PADIR—Port A data direction register	16 bits	33.2.1.3/-4
952	PAPAR—Port A pin assignment register	16 bits	33.2.1.4/-5

Table 2-1. MPC855T Internal Memory Map (continued)

Offset	Name	Size	Section/Page
954	PAODR—Port A open drain register	16 bits	33.2.1.1/-3
956	PADAT—Port A data register	16 bits	33.2.1.2/-4
958–95F	Reserved	8 bytes	—
960	PCDIR—Port C data direction register	16 bits	33.4.2.2/-15
962	PCPAR—Port C pin assignment register	16 bits	33.4.2.3/-15
964	PCSO—Port C special options register	16 bits	33.4.2.4/-16
966	PCDAT—Port C data register	16 bits	33.4.2.1/-14
968	PCINT—Port C interrupt control register	16 bits	33.4.2.5/-16
96A–96F	Reserved	6 bytes	—
970	PDDIR—Port D data direction register	16 bits	33.5.1.2/-18
972	PDPAR—Port D pin assignment register	16 bits	33.5.2/-19
974	Reserved	2 bytes	—
976	PDDAT—Port D data register	16 bits	33.5.1.1/-18
978–97F	Reserved	8 bytes	—
CPM General-Purpose Timers			
980	TGCR—Timer global configuration register	16 bits	17.2.3.1/-8
982–98F	Reserved	14 bytes	—
990	TMR1—Timer 1 mode register	16 bits	17.2.4/-9
992	TMR2—Timer 2 mode register	16 bits	17.2.4/-9
994	TRR1—Timer 1 reference register	16 bits	17.2.4.1/-10
996	TRR2—Timer 2 reference register	16 bits	17.2.4.1/-10
998	TCR1—Timer 1 capture register	16 bits	17.2.4.2/-10
99A	TCR2—Timer 2 capture register	16 bits	17.2.4.2/-10
99C	TCN1—Timer 1 counter	16 bits	17.2.4.3/-11
99E	TCN2—Timer 2 counter	16 bits	17.2.4.3/-11
9A0	TMR3—Timer 3 mode register	16 bits	17.2.4/-9
9A2	TMR4—Timer 4 mode register	16 bits	17.2.4/-9
9A4	TRR3—Timer 3 reference register	16 bits	17.2.4.1/-10
9A6	TRR4—Timer 4 reference register	16 bits	17.2.4.1/-10
9A8	TCR3—Timer 3 capture register	16 bits	17.2.4.2/-10
9AA	TCR4—Timer 4 capture register	16 bits	17.2.4.2/-10
9AC	TCN3—Timer 3 counter	16 bits	17.2.4.3/-11
9AE	TCN4—Timer 4 counter	16 bits	17.2.4.3/-11

Table 2-1. MPC855T Internal Memory Map (continued)

Offset	Name	Size	Section/Page
9B0	TER1—Timer 1 event register	16 bits	17.2.4.4/-11
9B2	TER2—Timer 2 event register	16 bits	17.2.4.4/-11
9B4	TER3—Timer 3 event register	16 bits	17.2.4.4/-11
9B6	TER4—Timer 4 event register	16 bits	17.2.4.4/-11
9B8–9BF	Reserved	8 bytes	—
Communications Processor (CP)			
9C0	CPCR—CP command register	16 bits	18.5.3/-6
9C2–9C3	Reserved	2 bytes	—
9C4	RCCR—RISC controller configuration register	16 bits	18.5.1/-4
9C6	Reserved	8 bits	—
9C7	RMDS—RISC microcode development support control register	8 bits	18.5.2/-5
9C8–9CB	Reserved	4 bytes	—
9CC	RCTR1—RISC controller trap register 1	16 bits	Used only by optional RAM microcode
9CE	RCTR2—RISC controller trap register 2	16 bits	Used only by optional RAM microcode
9D0	RCTR3—RISC controller trap register 3	16 bits	Used only by optional RAM microcode
9D2	RCTR4—RISC controller trap register 4	16 bits	Used only by optional RAM microcode
9D4–9D5	Reserved	2 bytes	—
9D6	RTER—RISC timer event register	16 bits	18.7.4/-15
9D8–9D9	Reserved	2 bytes	—
9DA	RTMR—RISC timers mask register	16 bits	18.7.4/-15
9DC–9EF	Reserved	20 bytes	—
Baud Rate Generators			
9F0	BRGC1—BRG1 configuration register	32 bits	20.4.1/-25
9F4	BRGC2—BRG2 configuration register	32 bits	20.4.1/-25
9F8	BRGC3—BRG3 configuration register	32 bits	20.4.1/-25
9FC	BRGC4—BRG4 configuration register	32 bits	20.4.1/-25
Serial Communications Controller 1 (SCC1)			
A00	GSMR_L1—SCC1 general mode register	32 bits	21.2.1/-3
A04	GSMR_H1—SCC1 general mode register	32 bits	21.2.1/-3

Table 2-1. MPC855T Internal Memory Map (continued)

Offset	Name	Size	Section/Page
A08	PSMR1—SCC1 protocol specific mode register	16 bits	21.2.2/-10 22.16/-13 (UART) 25.13.3/-11 (Asynchronous HDLC) 26.11/-10 (BiSYNC) 27.18/-19 (Ethernet) 28.9/-9 (Transparent)
A0A–A0B	Reserved	2 bytes	—
A0C	TODR1—SCC1 transmit-on-demand register	16 bits	21.2.4/-10
A0E	DSR1—SCC1 data synchronization register	16 bits	21.2.3/-10
A10	SCCE1—SCC1 event register	16 bits	22.19/-20 (UART) 23.11/-13 (HDLC) 25.13.1/-9 (Asynchronous HDLC) 26.14/-15 (BiSYNC) 27.21/-25 (Ethernet) 28.12/-12 (Transparent)
A12–A13	Reserved	2 bytes	
A14	SCCM1—SCC1 mask register	16 bits	
A16	Reserved	1 byte	—
A17	SCCS1—SCC1 status register	8 bits	22.20/-22 (UART) 23.12/-15 (HDLC) 25.13.2/-10 (Asynchronous HDLC) 26.15/-16 (BiSYNC) 28.13/-13 (Transparent)
A18–A1F	Reserved	8 bytes	—
Serial Management Controller 1 (SMC1)			
A82	SMCMR1—SMC1 mode register	16 bits	29.2.1/-3
A84–A85	Reserved	2 bytes	—
A86	SMCE1—SMC1 event register	8 bits	29.3.12/-20 (UART) 29.4.11/-31 (Transparent) 29.5.9/-39 (GCI)
A87–A89	Reserved	3 bytes	—
A8A	SMCM1—SMC1 mask register	8 bits	29.3.12/-20 (UART) 29.4.11/-31 (Transparent) 29.5.9/-39 (GCI)
A8B–A91	Reserved	7 bytes	—
Serial Management Controller 2 (SMC2)			
A92	SMCMR2—SMC2 mode register	16 bits	29.2.1/-3
A94–A95	Reserved	2 bytes	—
A96	SMCE2—SMC2 event register	8 bits	29.3.12/-20 (UART) 29.4.11/-31 (Transparent) 29.5.9/-39 (GCI)
A97–A99	Reserved	3 bytes	—

Table 2-1. MPC855T Internal Memory Map (continued)

Offset	Name	Size	Section/Page
A9A	SMCM2—SMC2 mask register	8 bits	29.3.12/-20 (UART) 29.4.11/-31 (Transparent) 29.5.9/-39 (GCI)
A9B–A9F	Reserved	5 bytes	—
Serial Peripheral Interface (SPI)			
AA0	SPMODE—SPI mode register	16 bits	30.4.1/-7
AA2–AA5	Reserved	4 bytes	—
AA6	SPIE—SPI event register	8 bits	30.4.2/-10
AA7–AA9	Reserved	3 bytes	—
AAA	SPIM—SPI mask register	8 bits	30.4.2/-10
AAB–AAC	Reserved	2 bytes	—
AAD	SPCOM—SPI command register	8 bits	30.4.3/-11
AAE–AB	Reserved	4 bytes	—
Parallel Interface Port (PIP) and Port B			
AB2	PIPC—PIP configuration register	16 bits	32.4.1/-8
AB4–AB5	Reserved	2 bytes	—
AB6	PTPR—PIP timing parameters register	16 bits	32.4.4/-10
AB8	PBDIR—Port B data direction register	32 bits	33.3.1.3/-10
ABC	PBPAR—Port B pin assignment register	32 bits	33.3.1.4/-11
AC0	PBODR—Port B open drain register	32 bits	33.3.1.1/-9
AC4	PBDAT—Port B data register	32 bits	33.3.1.2/-9
AC8–ADF	Reserved	24 bytes	—
Serial Interface (SI)			
AE0	SIMODE—SI mode register	32 bits	20.2.4.2/-13
AE4	SIGMR—SI global mode register	8 bits	20.2.4.1/-12
AE5	Reserved	8 bits	—
AE6	SISTR—SI status register	8 bits	20.2.4.5/-20
AE7	SICMR—SI command register	8 bits	20.2.4.4/-19
AE8–AEB	Reserved	4 bytes	—
AEC	SICR—SI clock route register	32 bits	20.2.4.3/-18
AF0	SIRP—Serial interface RAM pointer register	32 bits	20.2.4.6/-21
AF4–BFF	Reserved	268 bytes	—
C00–DFF	SIRAM—SI routing RAM	512 bytes	20.2.3.5/-10

Table 2-1. MPC855T Internal Memory Map (continued)

Offset	Name	Size	Section/Page
E00-1FFF	Reserved	4,608 bytes	—
Fast Ethernet Controller (FEC)			
E00	ADDR_LOW register	32 bits	43.4.1.1/-15
E04	ADDR_HIGH	32 bits	43.4.1.2/-15
E08	HASH_TABLE_HIGH	32 bits	43.4.1.3/-16
E0C	HASH_TABLE_LOW	32 bits	43.4.1.4/-17
E10	R_DES_START	32 bits	43.4.1.5/-17
E14	X_DES_START	32 bits	43.4.1.6/-18
E18	R_BUFF_SIZE	32 bits	43.4.1.7/-19
E40	ECNTRL	32 bits	43.4.1.8/-19
E44	IEVENT	32 bits	43.4.1.9/-20
E48	IMASK	32 bits	43.4.1.9/-20
E4C	IVEC	32 bits	43.4.1.10/-22
E50	R_DES_ACTIVE	32 bits	43.4.1.11/-22
E54	X_DES_ACTIVE	32 bits	43.4.1.12/-23
E80	MII_DATA	32 bits	/-24
E84	MII_SPEED	32 bits	43.4.1.14/-26
ECC	R_BOUND	32 bits	43.4.1.15/-27
ED0	R_FSTART	32 bits	43.4.1.16/-28
EE4	X_WMRK	32 bits	43.4.1.17/-29
EEC	X_FSTART	32 bits	43.4.1.18/-30
F34	FUN_CODE	32 bits	43.4.1.19/-31
F44	R_CNTRL	32 bits	43.4.1.20/-31
F48	R_HASH	32 bits	43.4.1.21/-32
F84	X_CNTRL	32 bits	43.4.1.22/-33
F88-1FFF	Reserved	4,215 bytes	
Dual-Port RAM (DPRAM)			
2000–2FFF	Dual-port system RAM	4,096 bytes	18.6.1/-10
3000–3BFF	Dual-port system RAM expansion	3,072 bytes	18.6.1/-10
3C00–3FFF	PRAM—Dual-port parameter RAM	1,024 bytes	18.6.3/-11

Part II
MPC8xx Microprocessor Module

Intended Audience

Part II is intended for users who need to understand the programming model of the embedded microprocessor. It assumes some familiarity with RISC architectures.

Contents

Part II describes the MPC8xx microprocessor embedded in the MPC855T. It provides detailed information on the registers and instructions that are implemented, the memory management unit (MMU), cache model, exception model, and an overview of instruction timing. It contains the following chapters:

- Chapter 3, “The MPC8xx Core,” provides an overview of the MPC855T core, summarizing topics described in further detail in subsequent chapters in Part II.
- Chapter 4, “MPC8xx Core Register Set,” describes the hardware registers accessible to the MPC855T core. These include both architecturally-defined and MPC855T-specific registers.
- Chapter 5, “MPC855T Instruction Set,” describes the instructions implemented by the MPC855T. These instructions are organized by the level of architecture in which they are implemented—UISA, VEA, and OEA.
- Chapter 6, “Exceptions,” describes the exception model implemented on the MPC855T.
- Chapter 7, “Instruction and Data Caches,” describes the organization of the on-chip instruction and data caches, cache control, various cache operations, and the interaction between the caches, the load/store unit (LSU), the instruction sequencer, and the system interface unit (SIU).
- Chapter 8, “Memory Management Unit” describes how the MMU is implemented on the MPC855T. Although the MPC855T MMU is based on the PowerPC MMU model, it differs greatly in many respects, which are described in this chapter.

- Chapter 9, “Instruction Execution Timing,” describes the MPC855T instruction unit, and provides ways to make greatest advantage of its RISC architecture characteristics, such as pipelining and parallel execution. It includes a table of instruction latencies and lists dependencies and potential bottlenecks.

Suggested Reading

This section lists additional reading that provides background for the information in this manual.

MPC8xx Documentation

Supporting documentation for the MPC855T can be accessed through the world-wide web at http://www.motorola.com. This documentation includes technical specifications, reference materials, and detailed application notes.

Related Documentation

The documentation is organized in the following types of documents:

- *Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture* (MPEFPC32B/AD)—Describes resources defined by the PowerPC architecture.
- User’s manuals—These books provide details about individual implementations and are intended for use with the *Programming Environments Manual*.
- Addenda/errata to user’s manuals—Because some processors have follow-on parts an addendum is provided that describes the additional features and functionality changes. These addenda are intended for use with the corresponding user’s manuals.
- Hardware specifications—Hardware specifications provide specific data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics, as well as other design considerations. Separate hardware specifications are provided for each part described in this book.
- Technical summaries—Each device has a technical summary that provides an overview of its features. This document is roughly the equivalent to the overview (Chapter 1) of an implementation’s user’s manual.
- *The Programmer’s Reference Guide for the PowerPC Architecture*: MPCPRG/D—This concise reference includes the register summary, memory control model, exception vectors, and the PowerPC instruction set.
- *The Programmer’s Pocket Reference Guide for the PowerPC Architecture*: MPCPRGREF/D—This foldout card provides an overview of PowerPC registers, instructions, and exceptions for 32-bit implementations.
- Application notes—These short documents address specific design issues useful to programmers and engineers working with Motorola processors.

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

Conventions

This chapter uses the following notational conventions:

Bold	Bold entries in figures and tables showing registers and parameter RAM should be initialized by the user.
mnemonics	Instruction mnemonics are shown in lowercase bold.
italics	Italics indicate variable command parameters, for example, **bcctr***x*. Book titles in text are set in italics.
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote binary number
rA, **r**B	Instruction syntax used to identify a source GPR
rD	Instruction syntax used to identify a destination GPR
REG[FIELD]	Abbreviations or acronyms for registers or buffer descriptors are shown in uppercase text. Specific bits, fields, or numerical ranges appear in brackets. For example, MSR[LE] refers to the little-endian mode enable bit in the machine state register.
x	In certain contexts, such as in a signal encoding or a bit field, indicates a don't care.
n	Indicates an undefined numerical value
¬	NOT logical operator
&	AND logical operator
|	OR logical operator

Acronyms and Abbreviations

Table i contains acronyms and abbreviations that are used in this document. Note that the meanings for some acronyms (such as SDR1 and DSISR) are historical, and the words for which an acronym stands may not be intuitively obvious.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
ALU	Arithmetic logic unit
BIST	Built-in self test
BPU	Branch processing unit
BUID	Bus unit ID

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
CR	Condition register
CRC	Cyclic redundancy check
CTR	Count register
DABR	Data address breakpoint register
DAR	Data address register
DEC	Decrementer register
DMA	Direct memory access
DRAM	Dynamic random access memory
DSISR	Register used for determining the source of a DSI exception
DTLB	Data translation lookaside buffer
EA	Effective address
FPR	Floating-point register
FPSCR	Floating-point status and control register
GPR	General-purpose register
IEEE	Institute of Electrical and Electronics Engineers
ITLB	Instruction translation lookaside buffer
IU	Integer unit
LIFO	Last-in-first-out
LR	Link register
LRU	Least recently used
LSB	Least-significant byte
lsb	Least-significant bit
LSU	Load/store unit
MMU	Memory management unit
MSB	Most-significant byte
msb	Most-significant bit
MSR	Machine state register
NaN	Not a number
No-op	No operation
OEA	Operating environment architecture
PCI	Peripheral component interconnect
PVR	Processor version register
RISC	Reduced instruction set computing

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
RTOS	Real-time operating system
RWITM	Read with intent to modify
Rx	Receive
SIMM	Signed immediate value
SPR	Special-purpose register
SPRG*n*	Registers available for general purposes
SR	Segment register
SRR0	Machine status save/restore register 0
SRR1	Machine status save/restore register 1
TB	Time base register
TLB	Translation lookaside buffer
Tx	Transmit
UIMM	Unsigned immediate value
UISA	User instruction set architecture
VA	Virtual address
VEA	Virtual environment architecture
XER	Register used primarily for indicating conditions such as carries and overflows for integer operations

Architecture Terminology Conventions

Table ii lists certain terms used in this manual that differ from the architecture terminology conventions.

Table ii. Terminology Conventions

The Architecture Specification	This Manual
Data storage interrupt (DSI)	DSI exception
Extended mnemonics	Simplified mnemonics
Instruction storage interrupt (ISI)	ISI exception
Interrupt	Exception
Privileged mode (or privileged state)	Supervisor-level privilege
Problem mode (or problem state)	User-level privilege
Real address	Physical address
Relocation	Translation
Storage (locations)	Memory
Storage (the act of)	Access

Table iii describes instruction field notation conventions used in this manual.

Table iii. Instruction Field Conventions

The Architecture Specification	Equivalent to:
BA, BB, BT	**crb**A, **crb**B, **crb**D (respectively)
BF, BFA	**crf**D, **crf**S (respectively)
D	d
DS	ds
FLM	FM
FXM	CRM
RA, RB, RT, RS	**r**A, **r**B, **r**D, **r**S (respectively)
SI	SIMM
U	IMM
UI	UIMM
/, //, ///	0...0 (shaded)

Chapter 3
The MPC8xx Core

This chapter provides an overview of the MPC8xx core, summarizing topics described in further detail in subsequent chapters in Part II. This chapter describes the functional specifications of the core. It is based on the *Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture*, which provides a more in-depth discussion of issues related to the 32-bit portion of the PowerPC architecture implementation.

The subset of PowerPC instructions supported by the MPC855T are listed in Chapter 5, “MPC855T Instruction Set.”

3.1 The MPC855T Core as a PowerPC Implementation

The core implements all PowerPC user-level instructions defined for 32-bit implementations except floating-point instructions (load/store and arithmetic). Likewise, it supports the registers defined by the PowerPC architecture necessary for the supported instructions.

The MPC855T core adheres to portions of the PowerPC architecture definition for supervisor operations. For example, it implements the PowerPC exception model (excluding inappropriate exceptions, such as those that support floating-point operations). The architecture-defined memory management model has been modified to suit the specific needs of the MPC855T core. Additional exceptions are defined (as permitted by the architecture) to support address translation.

The PowerPC architecture defines features not supported on the MPC855T hardware. These features include support for 64-bit addressing, multiprocessing, floating-point arithmetic, and some memory management features.

The core also implements MPC855T-specific development support features such as breakpoint and watchpoint mechanisms, program-flow tracking data generation, and debug mode operation.

3.2 PowerPC Architecture Overview

The PowerPC architecture takes advantage of recent technological advances in such areas as process technology, compiler design, and reduced instruction set computing (RISC)

microprocessor design to provide software compatibility across a diverse family of implementations, primarily single-chip microprocessors, intended for a wide range of systems, including battery-powered personal computers; embedded controllers; high-end scientific and graphics workstations; and multiprocessing, microprocessor-based mainframes.

To provide a single architecture for such a broad assortment of processor environments, the PowerPC architecture is both flexible and scalable.

The flexibility of the PowerPC architecture offers many price/performance options. Designers can choose whether to implement architecturally-defined features in hardware or in software. For example, a processor designed for a high-end workstation has greater need for the performance gained from implementing floating-point normalization and denormalization in hardware than a device using a PowerPC embedded controller might.

The PowerPC architecture defines the following features:

- Separate 32-entry register files for integer instructions. The general-purpose registers (GPRs) hold source data for integer arithmetic instructions.
- Instructions for loading and storing data between the memory system and the GPRs
- Uniform-length instructions to allow simplified instruction pipelining and parallel processing instruction dispatch mechanisms
- Non-destructive use of registers for arithmetic instructions in which the second, third, and sometimes the fourth operand, typically specify source registers for calculations whose results are typically stored in the target register specified by the first operand.
- A precise exception model
- A flexible architecture definition that allows certain features to be performed in either hardware or with assistance from implementation-specific software depending on the needs of the processor design
- User-level instructions for explicitly storing, flushing, and invalidating data in the on-chip caches. The architecture also defines special instructions (cache block touch instructions) for speculatively loading data before it is needed, reducing the effect of memory latency.
- A memory model that allows weakly-ordered memory accesses. This allows bus operations to be reordered dynamically, which improves overall performance and in particular reduces the effect of memory latency on instruction throughput.
- Support for separate instruction and data caches (Harvard architecture) and for unified caches
- Support for both big- and little-endian addressing modes

- Support for 64-bit addressing. The architecture supports both 32-bit or 64-bit implementations. This document describes the 32-bit portion of the PowerPC architecture. For information about the 64-bit architecture, see *Programming Environments Manual for Implementations of the PowerPC Architecture*.

3.2.1 Levels of the PowerPC Architecture

The PowerPC architecture is defined in three levels that correspond to three programming environments, roughly described from the most general, user-level instruction set environment, to the more specific, operating environment.

This layering of the architecture provides flexibility, allowing degrees of software compatibility across a wide range of implementations. For example, an implementation such as an embedded controller may support the user instruction set, whereas it may be impractical for it to adhere to the memory management, exception, and cache models.

The three levels of the PowerPC architecture are defined as follows:

- PowerPC user instruction set architecture (UISA)—The UISA defines the level of the architecture to which user-level (referred to as problem state in the architecture specification) software should conform. The UISA defines the base user-level instruction set, user-level registers, data types, the exception model as seen by user programs, and the memory and programming models.
- PowerPC virtual environment architecture (VEA)—The VEA defines additional user-level functionality that falls outside typical user-level software requirements. The VEA describes the memory model for an environment in which multiple devices can access memory, defines aspects of the cache model, defines cache control instructions, and defines the time base facility from a user-level perspective.

  Implementations that conform to the PowerPC VEA also adhere to the UISA, but may not necessarily adhere to the OEA.
- PowerPC operating environment architecture (OEA)—The OEA defines supervisor-level (referred to as privileged state in the architecture specification) resources typically required by an operating system. The OEA defines the PowerPC memory management model, supervisor-level registers, synchronization requirements, and the exception model. The OEA also defines the time base feature from a supervisor-level perspective.

  Implementations that conform to the PowerPC OEA also conform to the PowerPC UISA and VEA.

  The MPC855T adheres to the OEA definition of the exception model and provides a subset of the memory management model. It includes OEA-defined registers and instructions for configuration and exception handling.

Implementations that adhere to the VEA level are guaranteed to adhere to the UISA level; likewise, implementations that conform to the OEA level are also guaranteed to conform to

the UISA and the VEA levels. For a more detailed discussion of the characteristics of the PowerPC architecture, see the *Programming Environments Manual for Implementations of the PowerPC Architecture.*

For details regarding the MPC8xx core as an implementation of the PowerPC architecture, see Section 3.7, “The MPC855T and Implementation of the PowerPC Architecture.”

3.3 Features

Figure 3-1 shows the basic features of the MPC855T.

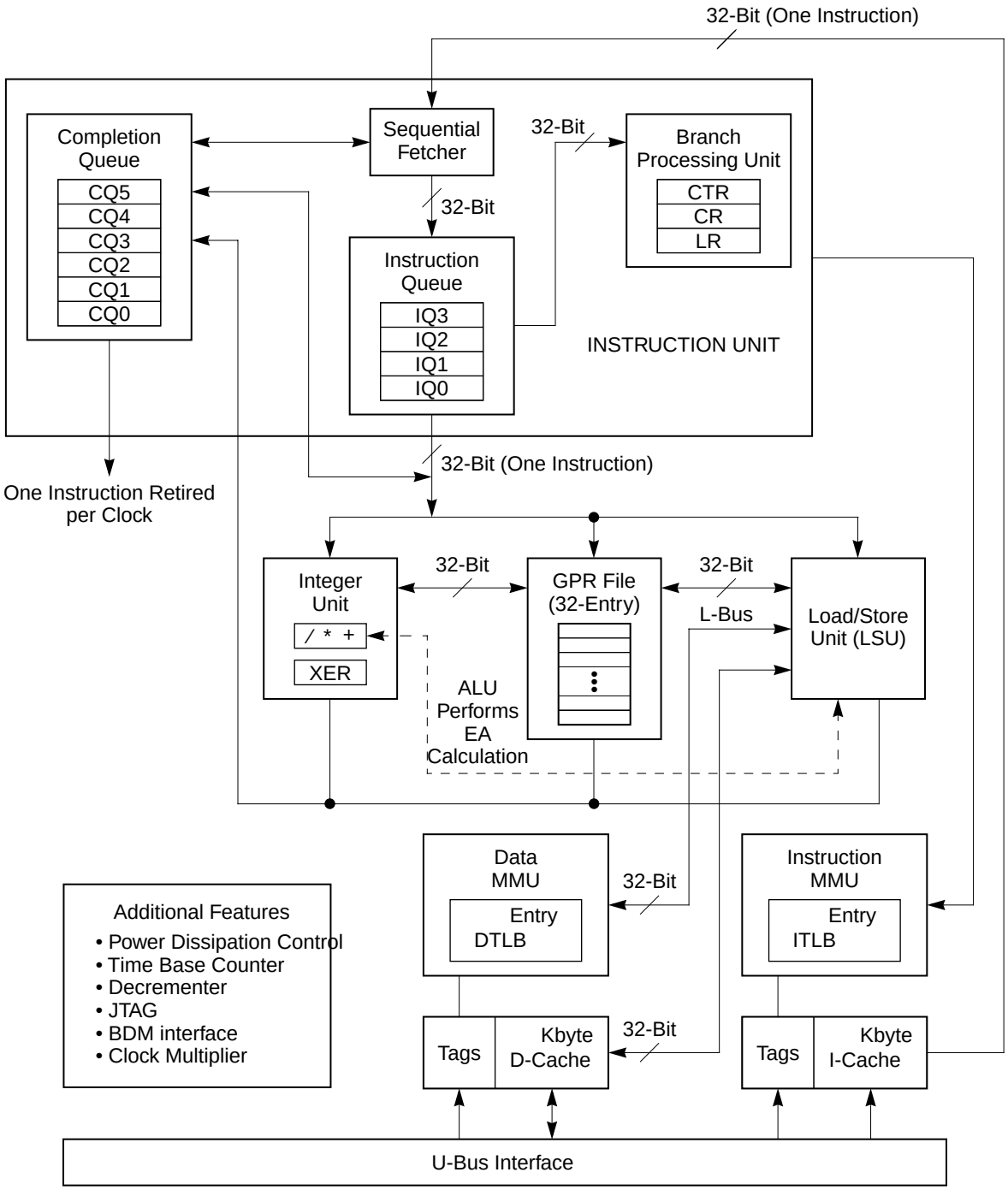


Figure 3-1. Block Diagram of the Core

The following is a list of the MPC8xx core main features:

- 32-bit implementation of PowerPC architecture features
  - — User-level instruction set (not including floating-point instructions)
  - — Thirty-two, 32-bit general-purpose registers (GPRs)

  - Registers required to support PowerPC user-level instruction set (except floating-point instructions). These include the integer exception register (XER), condition register (CR), link register (LR), and counter register (CTR).
  - Time base upper and time base lower and registers (TBU and TBL)
  - A subset of the supervisor-level registers for compliance with the following PowerPC models:
    - Configuration—Machine state register (MSR)
    - Exception model—Save/restore registers 0 and 1 (SRR0 and SRR1), DSI status register (DSISR), data address register (DAR)
  - Core-specific registers compliant with PowerPC architecture
  - Static branch prediction
  - Precise exception model that includes the subset of the PowerPC exceptions which supports the instruction set and memory management. The MPC855T implements all PowerPC asynchronous exceptions (interrupts)—system reset, machine check, decrementer, and external interrupts. MPC855T-specific exceptions are PowerPC-compliant.
  - Separate 32-entry instruction and data translation lookaside buffers (TLBs)
- Core-specific features
  - Fully static design
  - Additional registers that support the MPC855T-specific features
  - The ability to optimally issue and retire one instruction per clock cycle
  - Out-of-order execution and in-order completion
  - Extensive debug/testing support

3.4 Basic Structure of the Core

The MPC855T core consists of the following subunits:

- Instruction unit (sequencer)—Consists of the branch processing unit (BPU), the instruction queue, and the exception handling mechanism.
- Execution units—These consist of the following:
  - Integer unit—Implements all integer arithmetic and logical instructions defined by the PowerPC architecture:
  - Load/store unit (LSU)—Implements all load and store instructions except floating-point load/store instructions. Note that because the MPC855T does not implement floating-point load and store instructions, this document refers to integer load/store instructions simply as load/store instructions.

3.4.1 Instruction Flow

As many as one instruction per clock cycle is fetched into the four-entry instruction queue (IQ). The branch processing unit (BPU) predicts the outcome of branch instructions and in some cases, resolves whether the branch is taken. Figure 3-2 shows general instruction flow.

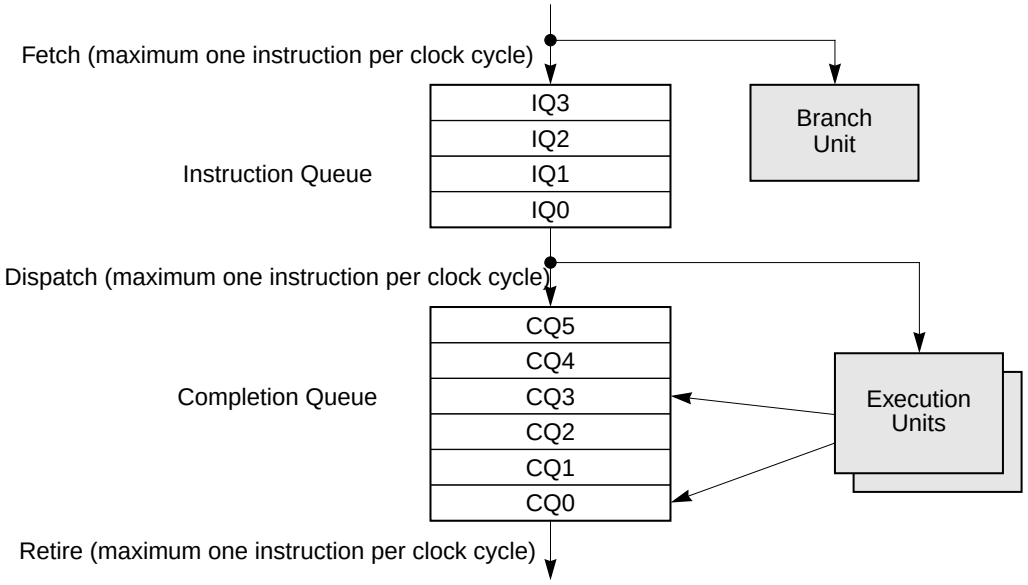


Figure 3-2. Instruction Flow Conceptual Diagram

Non-branch instructions reaching IQ0 are dispatched to the execution units at an optimal rate of one instruction per clock cycle. An instruction cannot be dispatched unless it can also take a position in the six-entry completion queue (CQ).

All branch instructions, including unconditional branch instructions, reaching IQ0 must also take a position in the completion queue. This allows program order to be maintained, it ensures a precise execution model, and it allows branch instructions to be used as breakpoints.

All instructions enter the CQ along with processor state information that can be affected by the instruction's execution. Executed arithmetic instructions pass their results both to rename buffers and to the architected registers (typically GPRs), but to ensure program order, instructions remain in the CQ until they can be retired.

If an exception occurs before the instruction can be retired, any results are removed from the rename buffer and GPR and the instruction is flushed from the completion queue, along with subsequent instructions that have not executed or have not dispatched.

This information is used to enable out-of-order completion of instructions and ensure a precise exception model. An instruction can be retired after all instructions ahead of it have retired and it updates the architected destination registers without taking an exception.

3.4.2 Basic Instruction Pipeline

Figure 3-3 shows instruction pipeline timing, showing how by distributing the processes required to fetch, execute, and retire an instruction into stages, multiple instructions can be processed during a single clock cycle.

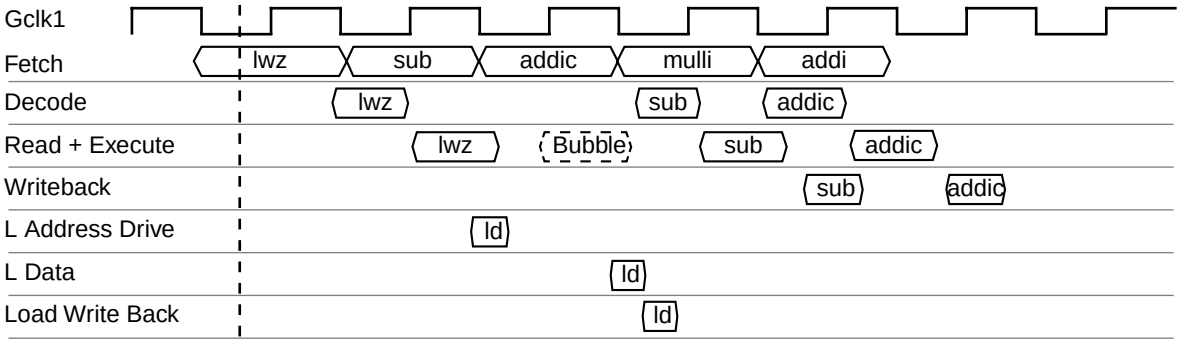


Figure 3-3. Basic Instruction Pipeline Timing

3.4.3 Instruction Unit

The instruction unit implements the basic instruction pipeline, fetches instructions from the memory system, dispatches them to available execution units, and maintains a state history to ensure a precise exception model and that operations finish in order. The instruction unit implements all branch processor instructions, including flow control and CR instructions. Table 9-1. describes instruction latencies.

3.4.3.1 Branch Operations

Because branch instructions can change program flow and because most branches cannot be resolved at the same time they are fetched, program branching can keep a processor from operating at maximum instruction throughput.

If a branch is mispredicted, additional time is required to flush the incorrect branch instructions and begin fetching from the correct target stream, which can create bubbles in the pipeline. To reduce the latency caused by misprediction, branch instructions allow the programmer to indicate whether a branch is likely to be taken. This is called static branch prediction.

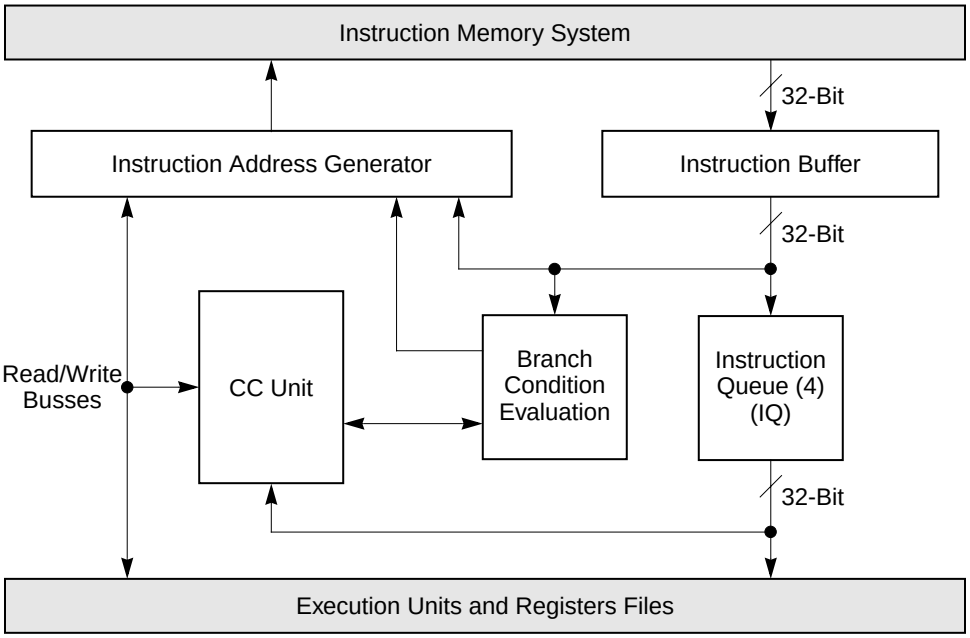


Figure 3-4. Sequencer Data Path

The instruction unit executes branches in parallel with those instructions that must be dispatched to an execution unit. Ideally, a instruction is dispatched to an execution unit every clock cycle, even when branches are in the code. The IQ also eliminates stalls due to instruction fetches that miss in the instruction cache or that generate a page fault. All instructions are fetched into the IQ, and all instructions except branch instructions are dispatched to the execution units when they reach IQ0. Branches enter the queue to mark watchpoints. See Chapter 44, “System Development and Debugging.” Because branches do not prevent the issue of nonbranch instructions unless they come in pairs, the performance impact of entering branches in the IQ is negligible.

The core also implements a branch reservation station and static branch prediction so branches can be resolved as early as possible. The reservation station allows a branch instruction to pass from the IQ before its condition is ready. With the branch out of the way, fetching can continue as the branch is evaluated. Static branch prediction (defined by the PowerPC UISA) determines which instruction stream is prefetched while the branch is being resolved. When the branch operand becomes available, it is forwarded to the BPU and the condition is evaluated. The static branch prediction mechanism is shown in Table 3-1.

Table 3-1. Static Branch Prediction

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 (LR or CTR) address ready	Fall through	Taken

Table 3-1. Static Branch Prediction (continued)

Branch Type	Default Prediction (y=0)	Modified Prediction (y=1)
BCLR or BCCTR (LR or CTR) address not ready	Wait	Wait
B (unconditional branch)	Taken	Taken

Branch instructions whose condition is unavailable are issued to the reservation station until they are predicted. Branch instructions that issue with source data already available do not require prediction (and are said to be resolved). Instructions fetched under a predicted branch are conditionally fetched. The core flushes instructions conditionally fetched under a mispredicted branch.

3.4.3.2 Dispatching Instructions

The sequencer can dispatch a sequential instruction on each clock if the appropriate execution unit is available and a position is free in the completion queue. The execution unit must be able to discern whether source data is available and to ensure that no other executing instruction targets the same destination register. The sequencer informs the execution units of the existence of the instruction on the instruction bus. The execution units decode the instruction, check whether the source and destination operands are free, and inform the sequencer whether instructions can be dispatched.

3.5 Register Set

Registers implemented in the MPC855T core can be grouped as follows:

- PowerPC registers. The MPC855T implements the user registers defined by the UISA and VEA portions of the architecture except for those that support floating-point operations. PowerPC registers implemented on the MPC855T are described in Section Chapter 4, “MPC8xx Core Register Set,” and Section 4.1.2, “PowerPC Registers—Supervisor Registers.”
- Implementation-specific registers. These are all special-purpose registers (SPRs). These are described in Section 4.1.3, “MPC855T-Specific SPRs.”

3.6 Execution Units

As shown in Figure 3-1, the MPC855T allows parallel execution of instructions using separate branch processing unit (BPU), load/store unit (LSU), and integer unit (IU). These execution units are described in the following sections.

3.6.1 Branch Processing Unit

The branch processing unit differs from the other execution units in that it examines branch instructions while they are in the IQ. Other instructions are dispatched to the IU and LSU from IQ0. For details about the performance of various instructions, see Table 3-1.

The core supports the UISA-defined static branch prediction. That is, the y bit is used to provide a hint as to whether the branch the branch is likely to be taken or not taken. No prediction is done for branches to the link register or count register if the target address is not ready (see Table 3-1 for details).

3.6.2 Integer Unit

The core implements the following types of integer instructions:

- Arithmetic instructions
- Compare instructions
- Trap instructions
- Logical instructions
- Rotate and shift instructions

Most integer instructions can execute in one clock cycle. For details about the performance of the various instructions, see Table 3-1 of this manual.

Note the following special cases:

- If an **mtspr** or **mfspr** instruction specifies an invalid SPR in which **spr[0]** = 1, a program exception occurs if the processor is in user mode. Valid SPRs are listed in Chapter 4, “MPC8xx Core Register Set.”
- If **divw[o][.]** is used to perform either (0x80000000 ÷ -1) or (<anything> ÷ 0), the contents of **r**D are 0x8000_0000 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 MPC855T, 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.

3.6.3 Load/Store Unit

The load/store unit (LSU) transfers all data between the GPRs and the processor’s internal bus. It is implemented as an independent execution unit so that stalls in the memory pipeline affect the master instruction pipeline only if there is a data dependency.

The following lists the LSU’s main features:

- All instructions implemented in hardware, including unaligned, string, and multiple accesses
- Two-entry load/store instruction address queue
- Pipelined operation. The LSU pipelines load accesses. Individual cache accesses of all multiple-register instructions and unaligned accesses are pipelined into the data cache interface.
- Load/store multiple and string instructions synchronize
- Load/store breakpoint/watchpoint detection support
- The LSU implements cache and TLB management instructions as special bus write cycles, which are issued to the data cache interface.

Figure 3-5 is a block diagram of the LSU and its two queues. The address queue is a 2-entry queue shared by all load/store instructions and the integer data queue is a 2-entry, 32-bit queue that holds integer data.

The LSU has a dedicated writeback bus so that loaded data received from the internal bus is written directly back to the GPRs.

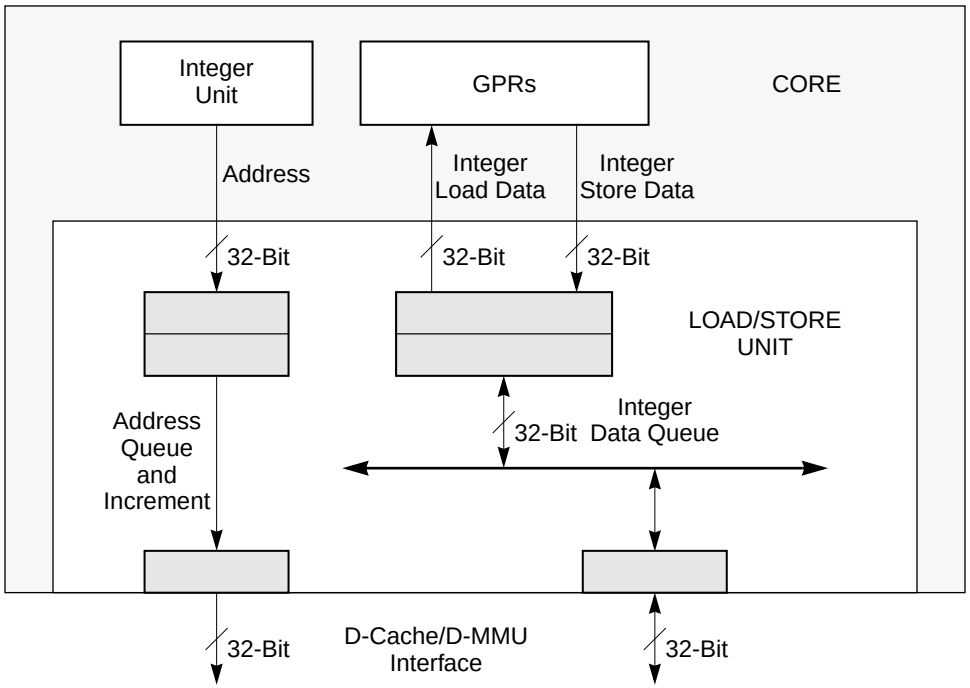


Figure 3-5. LSU Functional Block Diagram

To execute multiple/string instructions and unaligned accesses, the LSU increments the EA to access all necessary data. This allows the LSU to execute unaligned accesses without stalling the master instruction pipeline.

3.6.3.1 Executing Load/Store Instructions

When load or store instructions are dispatched, the LSU determines if all of the operands are available. These operands include the following:

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

If all operands are available, the LSU takes the instruction and enables the sequencer to issue a new instruction. Then, using a dedicated interface, the LSU notifies the integer unit of the need to calculate the EA. All load/store instructions are executed and finished in order. If no prior instructions are in the address queue, the load/store operation is issued to the data cache when the instruction executes. Otherwise, if prior instructions remain whose addresses have not been issued to the data cache, the instruction's address and data are placed in their respective queues. For load/store with update instructions, the destination address register is written back on the following clock cycle, regardless of the address queue's state.

3.6.3.2 Serializing Load/Store Instructions

The following load/store instructions are not executed until all previous instructions have finished.

- Load/store multiple instructions—**lmw**, **stmw**
- Memory synchronization instructions—**lwarx**, **stwcx.**, **sync**
- String instructions—**lswi**, **lswx**, **stswi**, **stswx**
- Move to SPRs

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

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

3.6.3.3 Store Accesses

Because the core supports the precise exception model, a new store instruction cannot update the data cache until all prior instructions have finished without an exception. If a store instruction follows a load instruction, a one-clock delay is inserted between the load bus cycle termination and the store cycle issue.

3.6.3.4 Nonspeculative Load Instructions

Load instructions targeted at nonspeculative memory are identified as nonspeculative one clock cycle after the previous load/store bus cycle ends, only if all prior instructions have finished without an exception.

The nonspeculative identification relates to the state of the cycle's associated instruction. For **lmw**, although the accesses are pipelined into the bus, they are all marked as nonspeculative because the instruction is nonspeculative. If a single-register load instruction generates more than one bus cycle, some of the cycles can be marked as speculative and later cycles can be marked as nonspeculative after all prior instructions end. Speculative load accesses to external memory marked nonspeculative cannot occur until the load instruction becomes nonspeculative.

3.6.3.5 Unaligned Accesses

Although the 32-bit U-bus supports only naturally aligned transfers, the LSU supports unaligned accesses in hardware by breaking them into a pipelined series of aligned transfers. Table 3-2 shows the number of bus cycles needed for single-register load/store accesses.

Table 3-2. Bus Cycles Needed for Single-Register Load/Store Accesses

Transfer Size	Transfer Address (Last Two Bits)	Number of Bus Cycles	Transfer Type	Address/Size
Byte	0x00	1	Aligned	0x00/byte
	0x01	1	Aligned	0x01/byte
	0x02	1	Aligned	0x02/byte
	0x03	1	Aligned	0x03/byte
Half Word	0x00	1	Aligned	0x00/halfword
	0x01	2	Unaligned	0x01/byte 0x02/byte
	0x02	1	Aligned	0x02/halfword
	0x03	2	Unaligned	0x03/byte 0x04/byte
Word	0x00	1	Aligned	0x00/word
	0x01	3	Unaligned	0x01/byte 0x02/halfword 0x05/byte
	0x02	2	Unaligned	0x02/halfword 0x04/halfword
	0x03	3	Unaligned	0x03/byte 0x04/halfword 0x06/byte

3.6.3.6 Atomic Update Primitives

The **lwarx** and **stwcx**. instructions are atomic update primitives and are used to set and clear memory reservations. Reservation accesses made by the same processor are implemented by the LSU. The external bus interface implements memory reservations as they relate to accesses made by external bus devices. Accesses made by other internal devices to internal memories implement memory reservations as they relate to special internal bus snoop logic.

When an **lwarx** instruction executes, the LSU issues a cycle to the data cache with a special attribute. For external memory accesses, this attribute causes the external bus interface to set a memory reservation during the address tenure. External logic must then snoop the external bus to determine if another device breaks the memory reservation by accessing the same location. $\overline{KR}$ and $\overline{CR}$ signals are available to external logic to signal loss of a reservation to the external bus interface. When an **stwcx**. instruction addresses external memory and the external bus interface determines that the reservation was lost, it blocks the external bus access and notifies the LSU.

The MPC855T supports the memory reservation mechanism in a hierarchical bus structure. For reservations on internal memory, an **lwarx** causes on-chip snoop logic to latch the address. This logic notifies the LSU of any internal master store access and resets the reservation. If a new **lwarx** instruction address tenure executes successfully, it replaces any previous reservation address at the appropriate snoop logic. However, executing an **stwcx.** instruction cancels the reservation unless an alignment exception is detected.

3.7 The MPC855T and Implementation of the PowerPC Architecture

This section describes the relationship between the MPC855T and implementation of the PowerPC architecture. It indicates the types of distinguishing features of the MPC855T described in the following:

- In many cases, the PowerPC architecture specification is flexible enough to allow implementation options. For example, the architecture does not specify whether unaligned transfers must be handled in hardware or whether instruction execution must be performed in hardware or software.
- The PowerPC architecture defines optional features, some of which are implemented on the MPC855T (such as TLBs) and some of which are not, such as the **eciwx** and **ecowx** instructions.
- The PowerPC architecture defines features, such as virtual memory and floating-point instructions, that are not implemented on the MPC855T.

Table 3-3 summarizes MPC855T features with respect to the UISA definition.

Table 3-3. UISA-Level Features

Functionality	Description
Reserved fields	Reserved fields in instructions are described under the specific instruction definition in Chapter 5, “MPC855T Instruction Set.” Unless otherwise stated, instruction fields marked I, II, and III are discarded during decoding. Thus, this type of instruction yields results of the defined instructions with the appropriate field = 0. In most cases, reserved fields in registers are ignored on write and return zeros for them on read for any control register implemented by the core. Exceptions are XER[16–23] and the reserved bits of MSR, which are set by the source value on write and return the value last set for it on read.
Classes of Instructions	Required instructions (except floating-point load, store, and compute instructions) are implemented in hardware. Optional instructions are executed by implementation-dependent code; any attempt to execute one of these commands causes the core to take the software emulation exception (offset 0x01000). Illegal and reserved instruction class instructions are supported by implementation-dependent code and, thus the core hardware generates a software emulation exception.
Exceptions	Invocation of the system software for any exception caused by an instruction in the core is precise, regardless of the type and setting.
Fetching instructions	The core fetches a number of instructions into its IQ from which they are dispatched to the execution units. If a program modifies instructions, it should call a system library program to ensure that the instruction fetching mechanism can detect changes before execution.
Branch instructions	The core implements all UISA instructions defined for the branch processor in hardware. For details about the performance of various instructions, see Table 3-1.
Invalid branch instruction forms	Bits marked with z in the BO encoding definition default to z = 0 and are discarded by the core decoding. Thus, these instructions yield results of defined instructions for which z = 0. 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.
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 3-1.).
Integer processor	The core implements the following integer instructions: • Arithmetic instructions • Compare instructions • Trap instructions • Logical instructions • Rotate and shift instructions
Move to/from SPR instructions	Move to/from invalid SPRs in which SPR[0] = 1 invokes the privileged instruction error exception handler if the processor is in user mode.
Integer arithmetic instructions	Attempting to use **divw** to perform either 0x80000000 ÷ -1 or <anything> ÷ 0 sets the contents of **r**D to 0x80000000 and if Rc =1, the contents CR0 are LT = 1, GT = 0, and EQ = 0. 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 MPC855T, 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.
Integer load/store with update instructions	For load with update and store with update instructions where **r**A = 0, the EA is written into **r**0. For load with update instructions where **r**A = **r**D, **r**A is boundedly undefined.

Table 3-3. UISA-Level Features (continued)

Functionality	Description
Integer load/ 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 an **lmw** instruction (if **r**A is in the range of registers to be loaded), the instruction completes normally. **r**A is then loaded from the memory location as follows: **r**A <- MEM(EA+(**r**A-**r**D)*4, 4)
Integer load string instructions	Load string instructions behave like load multiple instructions with respect to invalid format in which **r**A is in the range of registers to be loaded. If **r**A is in the range, it is updated from memory.
Memory synchronization instructions	For these instructions, if EA is not a multiple of four, the system alignment error handler is invoked.
Optional instructions	No optional instructions are supported.
Little-endian byte ordering	The LSU supports little-endian byte ordering as specified in the UISA. In little-endian mode, trying to execute an unaligned individual scalar or multiple/string access causes an alignment exception.

Table 3-4 summarizes MPC855T features with respect to the VEA definition.

Table 3-4. VEA-Level Features

Functionality	Description
Memory coherency	Memory coherency is not supported in the MPC855T hardware, but can be performed in the software or by defining memory as cache inhibited. In addition, the MPC855T does not provide any data storage attributes to an external system.
Atomic update primitives	Both the **lwarx** and **stwcx.** instructions are implemented according to the PowerPC architecture requirements. When memory 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 memory region. Therefore, if a data cache miss occurs, the access on the internal and external buses does not have a reservation attribute. The MPC855TMPC855T does not cause the system DSI exception handler to be invoked if memory accessed by the **lwarx** and **stwcx.** instructions is in write-through required mode. Also, the MPC855T does not support snooping an external bus activity outside the chip. The provision is made to cancel the reservation inside the MPC855T by using the $\overline{CR}$ and $\overline{KR}$ input signals. For accesses to internal resources, internal snoop logic monitors the internal bus for communication processor module (CPM) accesses of the address associated with the last **lwarx** instruction.
The effect of operand placement on performance	The LSU hardware supports all PowerPC integer load/store instructions. Naturally-aligned operands give optimal performance for a maximum size of four bytes. Unaligned operands are supported in hardware and are broken into a series of aligned transfers. The effect of operand placement on performance is as stated in the VEA, except for 8-byte operands. Because the MPC855T uses a 32-bit data bus, performance is good rather than optimal. See Section 3.6.3.5, “Unaligned Accesses for a description of integer unaligned instruction execution and timing and to Section 9.2.2, “String Instruction Latency,” for a description of string instruction timing.

Table 3-4. VEA-Level Features (continued)

Functionality	Description
Memory control instructions	The MPC855T interprets cache control instructions as if they pertain only to the MPC855T cache. These instructions do not broadcast. Any bus activity caused by these instructions results from an operation performed on the MPC855T cache and not because of the instruction itself. • •Instruction Cache Block Invalidate (**icbi**)—The MMU translates the EA and the associated instruction cache block is invalidated if hit. • •Instruction Synchronize (**isync**)—The **isync** instruction waits for all previous instructions to complete and then discards any prefetched instructions, causing subsequent instructions to be fetched or refetched from memory and executed. • •Data Cache Block Touch (**dcbt**) and Data Cache Block Touch for Store (**dcbtst**)—The appropriate cache block is checked for a hit. If it is a miss, the instruction is treated as a regular miss, except that bus error does not cause an exception. If no error occurs, the cache is updated. • •Data Cache Block Set to Zero (**dcbz**)—Executes as defined in the VEA. • •Data Cache Block Store (**dcbst**)—Executes as defined in the VEA. • •Data Cache Block Invalidate (**dcbi**)—The MMU translates the EA and the associative data cache block is invalidated if hit. • •Data Cache Block Flush (**dcbf**)—Executes as defined in the VEA. • •Enforce In-Order Execution of I/O (**eieio**)—When executing an **eieio** instruction, the LSU waits for previous accesses to terminate before beginning accesses associated with load/store instructions after the **eieio** instruction.
Time base	The time base functions as defined by the VEA and supports an additional implementation-specific exception. The time base is described in Chapter 10, “System Interface Unit,” and in Chapter 14, “Clocks and Power Control.”

Table 3-5 summarizes MPC855T features with respect to the OEA definition.

Table 3-5. OEA-Level Features

Functionality	Description
Machine state register	The floating-point exception mode (bits FE0 and FE1) is ignored by the MPC855T. The IP bit initial state after reset is set as programmed by the reset configuration specified in Section 6.1.2.1, “System Reset Interrupt (0x00100).”
Processor version register	The value of the PVR register’s version field is 0x0050. The value of the revision field is 0x0000 and it is incremented each time the software distinguishes between the revisions.
Other OEA registers	The following registers are not implemented: SDR1, BAT registers, segment registers, and EAR
Page size	The MPC855T differs from the OEA-defined memory management mode with respect to page sizes. Page sizes are 4, 16, and 512 Kbytes, and 8 Mbytes with an optional subpage granularity of 1 Kbyte for 4-Kbyte pages in a maximum physical memory size of 4 Gbytes. Neither ordinary or direct-store segments are supported.
Address space	The MPC855T differs from the OEA-defined memory management model. Specifically, it does not support the same address translation mechanism that requires an intermediate 52-bit virtual address. It also does not support block address translation or the associated block address translation SPRs. In its place, the MPC855T’s internal memory space includes memory-mapped control registers and memory 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 device.

Table 3-5. OEA-Level Features (continued)

Functionality	Description
Address translation	If address translation is disabled (MSR[IR] = 0 for instruction accesses or MSR[DR] = 0 for data accesses), the EA is treated as the physical address and is passed directly to the memory subsystem. Otherwise, the EA is translated by using the MMU's TLB mechanism. Instructions are not fetched from no-execute or guarded memory and data accesses are not executed speculatively to or from the guarded memory. The features of the MMU hardware are as follows: • 8entry fully associative ITLB • 8-entry fully associative DTLB • Supports up to 16 virtual address spaces • Supports 16 access protection groups • Supports fast software table search mechanism The MPC855T MMU is described in detail in Chapter 8, "Memory Management Unit."
Reference and change bits	No reference bit is supported by the MPC855T. However, the change bit is supported by using the data TLB error exception mechanism when writing to an unmodified page.
Memory protection	Two protection modes are supported by the MPC855T: • Domain manager mode • PowerPC mode See Chapter 8, "Memory Management Unit."

Chapter 4
MPC8xx Core Register Set

This chapter describes the software-accessible registers implemented on the MPC855T. These include registers that are defined by the PowerPC architecture and registers that are specific to the MPC855T. This section does not include registers that are part of the communication processor module (CPM); these registers are described in Part V, "Communications Processor Module." Refer to the *Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture* for more information about the architecture's register definition.

4.1 MPC855T Register Implementation

Registers implemented in the MPC855T core can be grouped as follows:

- Two types of registers as defined by the PowerPC architecture.
  - — User registers, which can be accessed by user-level software. All PowerPC user-level registers are defined by the user instruction set architecture (UISA) except for the time base registers, which can be read by user-level software and are defined by the virtual environment architecture (VEA). User registers are described in Section 4.1.1, "PowerPC Registers—User Registers."
  - — Supervisor registers, which can be accessed by supervisor software and in some cases are the automatic result of hardware activity, such as when an exception is taken and when the system is reset. All supervisor registers are defined by the operating environment architecture (OEA), except the time base registers, which can be written to only by supervisor software and are defined by the VEA. PowerPC supervisor registers are described in Section 4.1.2, "PowerPC Registers—Supervisor Registers."

  The UISA, VEA, and OEA architecture definitions are described in Section 3.2.1, "Levels of the PowerPC Architecture."
- MPC855T-specific registers. These registers are either supervisor-level registers or debug registers. These are described briefly in Section 4.1.3, "MPC855T-Specific SPRs," Table 4-9 and Table 2-1 provide cross references to the sections in this book where each register is described.

4.1.1 PowerPC Registers—User Registers

The MPC855T implements the user-level registers defined by the PowerPC architecture except those required for supporting floating-point operations (the floating-point register file (FPRs) and the floating-point status and control register (FPSCR)). User-level, PowerPC registers are listed in Table 4-1 and Table 4-2. Table 4-2 lists user-level special-purpose registers (SPRs).

Table 4-1. User-Level PowerPC Registers

Description	Name	Comments	Access Level	Serialize Access
General-purpose registers	GPRs	The thirty-two 32-bit (GPRs) are used for source and destination operands.	User	—
Condition register	CR	See Section 4.1.1.1.1, “Condition Register (CR).”	User	Only **mtcrf**

Table 4-2 lists SPRs defined by the PowerPC architecture implemented on the MPC855T.

Table 4-2. User-Level PowerPC SPRs

SPR Number			Name	Comments	Serialize Access
Decimal	SPR [5–9]	SPR [0–4]			
1	00000	00001	XER	See Section 4.1.1.1.3, “XER.”	Write: Full sync Read: Sync relative to load/store operations
8	00000	01000	LR	See the *Programming Environments Manual*	No
9	00000	01001	CTR	See the *Programming Environments Manual*	No
268	01000	01100	TBL read [1]	Section 10.9, “The Timebase.”	Write (as a store)
269	01000	01101	TBU read [2]		

[1] Extended opcode for mftb, 371 rather than 339.

[2] Any write (**mtspr**) to this address causes an implementation-dependent software emulation exception.

4.1.1.1 PowerPC User-Level Register Bit Assignments

This section describes bit assignments of PowerPC registers implemented by the MPC855T. For more details, see the *Programming Environments Manual for 32-Bit Processors*.

4.1.1.1.1 Condition Register (CR)

The condition register (CR) is a 32-bit register that reflects the result of certain operations and provides a mechanism for testing and branching. The bits in the CR are grouped into eight 4-bit fields, CR0–CR7, as shown in Figure 4-1.

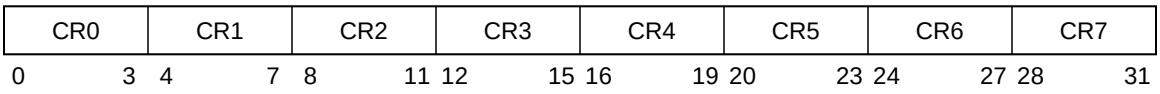


Figure 4-1. Condition Register (CR)

The CR fields can be set in one of the following ways:

- Specified fields of the CR can be set from a GPR by using the **mtcrf** instruction.
- An **mcrf** instruction can move the contents of XER[0–3] to a CR field.
- An **mcrxr** instruction can copy a specified XER field to a specified CR field.
- Condition register logical instructions perform logical operations on specified CR bits.
- CR0 can be the implicit result of an integer instruction.
- A specified CR field can indicate the result of an integer compare instruction.

Note that branch instructions are provided to test individual CR bits.

4.1.1.1.2 Condition Register CR0 Field Definition

For all integer instructions, when the CR is set to reflect the result of the operation (that is, when Rc = 1), and for **addic.**, **andi.**, and **andis.**, CR0[0–2] are set by an algebraic comparison of the result to zero; CR0[3] is copied from XER[SO]. For integer instructions, CR[0–3] reflects the result as a signed quantity.

The CR bits are interpreted as shown in Table 4-3. If any portion of the result is undefined, the value placed into CR0[0–3] is undefined.

Table 4-3. Bit Settings for CR0 Field of CR

CR0 Bit	Description
0	Negative (LT). Set when the result is negative.
1	Positive (GT). Set when the result is positive (and not zero).
2	Zero (EQ). Set when the result is zero.
3	Summary overflow (SO). This is a copy of the final state of XER[SO] at the completion of the instruction.

Note that CR0 may not reflect the true (that is, infinitely precise) result if overflow occurs.

4.1.1.1.3 XER

Figure 4-2 shows XER bit assignments. Settings are based on the final result produced by executing an instruction.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	SO	OV	CA	—												
Reset	0000_0000_0000_0000															
R/W	R/W															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—									BCNT						
Reset	0000_0000_0000_0000															
R/W	R/W															

Figure 4-2. XER Register

XER bits are described in Table 4-4.

Table 4-4. XER Field Definitions

Bit(s)	Name	Description
0	SO	Summary overflow. Set when an instruction (except **mtspr**) sets the overflow bit (OV). Once set, SO remains set until it is cleared by an **mtspr(XER)** or an **mcrxr** instruction. It is not altered by compare instructions or other instructions (except **mtspr(XER)** and **mcrxr**) that cannot overflow.
1	OV	Overflow. Set to indicate that an overflow occurred during execution of an instruction. Add, subtract from, and negate instructions with OE = 1 set OV if the carry out of the msb is not equal to the carry out of the msb + 1 and clear it otherwise. Multiply low and divide instructions with OE = 1 set OV if the result cannot be represented in 32 bits (**mullw**, **divw**, **divwu**) and clear it otherwise. The OV bit is not altered by compare instructions that cannot overflow (except **mtspr(XER)** and **mcrxr**).
2	CA	Carry. Set during execution of the following instructions: • •Add carrying, subtract from carrying, add extended, and subtract from extended instructions set CA if there is a carry out of the msb, and clear it otherwise. • • Shift right algebraic instructions set CA if any 1 bits have been shifted out of a negative operand, and clear it otherwise. The CA bit is not altered by compare instructions, nor by other instructions that cannot carry (except shift right algebraic, **mtspr(XER)**, and **mcrxr**).
3–24	—	Reserved
25–31	BCNT	Specifies the number of bytes to be transferred by a Load String Word Indexed (**lswx**) or Store String Word Indexed (**stswx**) instruction.

Although divide instructions have a relatively long latency, they can update XER[OV] 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.

4.1.1.1.4 Time Base Registers

The time base registers (TBU and TBL) are described in Section 10.9, “The Timebase,” and in Chapter 14, “Clocks and Power Control.” The PowerPC architecture does not define an exception associated directly with the time base, but one is implemented in the MPC855T.

4.1.2 PowerPC Registers—Supervisor Registers

All supervisor-level registers implemented on the MPC855T are SPRs, except for the machine state register (MSR), described in Table 4-5.

Table 4-5. Supervisor-Level PowerPC Registers

Description	Name	Comments	Serialize Access
Machine state register	MSR	See Section 4.1.2.3.1, “Machine State Register (MSR).”	Write fetch sync

Table 4-6 lists supervisor-level SPRs defined by the PowerPC architecture.

Table 4-6. Supervisor-Level PowerPC SPRs

SPR Number			Name	Comments	Serialize Access
Decimal	SPR[5–9]	SPR[0–4]			
18	00000	10010	DSISR	See the *Programming Environments Manual* and Section 4.1.2.1, “DAR, DSISR, and BAR Operation.”	Write: Full sync Read: Sync relative to load/store operations
19	00000	10011	DAR	See the *Programming Environments Manual* and Section 4.1.2.1, “DAR, DSISR, and BAR Operation.”	Write: Full sync Read: Sync relative to load/store operations
22	00000	10110	DEC	See Section 10.8.1, “Decrementer Register (DEC),” and in Chapter 14, “Clocks and Power Control”	Write
26	00000	11010	SRR0	See SRR0 settings for individual exceptions in Chapter 6, “Exceptions.”	Write
27	00000	11011	SRR1	See SRR1 settings for individual exceptions in Chapter 6, “Exceptions.”	Write
272	01000	10000	SPRG0	See the *Programming Environments Manual*.	Write
273	01000	10001	SPRG1		
274	01000	10010	SPRG2		
275	01000	10011	SPRG3		
284	01000	11100	TBL write [1]	See Section 10.9, “The Timebase,” and Chapter 14, “Clocks and Power Control.”	Write (as a store)
285	01000	11101	TBU write[1]		
287	01000	11111	PVR	Section 4.1.2.3.2, “Processor Version Register.”	No (read-only register)

[1] Any read (**mftb**) to this address causes an implementation-dependent software emulation exception.

4.1.2.1 DAR, DSISR, and BAR Operation

The LSU updates the DAR, DSISR, and BAR when an exception is taken.

- When a bus error occurs, the data address register (DAR) is loaded with the effective address. For instructions that generate multiple accesses, the effective address of the first offending tenure is loaded.
- The DSI status register (DSISR) notifies the error handler when an exception is caused by a load or store. For a data MMU error, the data MMU loads the DSISR with error status. For alignment exceptions, the DSISR is loaded with the instruction information as defined by the PowerPC architecture.
- The breakpoint address register (BAR) notifies the address on which a data breakpoint occurred. For a multiple-cycle 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 exception is taken. At any other time, its value is boundedly undefined (this term is defined very specifically by the PowerPC architecture and is discussed in the *Programming Environments Manual*).

The following situations cause the DAR, BAR, and DSISR registers to be updated.

Table 4-7. Value Summary of the DAR, BAR, and DSISR Registers

Exception Type	DAR Value	DSISR Value	BAR Value
DSI	Cycle EA	Data MMU error status	Undefined
Alignment	Data EA	Instruction information	Undefined
Data breakpoint	Does not change	Does not change	Cycle EA
Machine check	Cycle EA	Instruction information	Undefined
Software emulation exception	Does not change	Does not change	Undefined
Floating-point unavailable	Does not change	Does not change	Undefined
Program exception	Does not change	Does not change	Does not change

4.1.2.2 Unsupported Registers

The MPC855T does not support the following OEA registers:

- DBATs and IBATs —The MPC855T does not support block address translation.
- EAR—The MPC855T does not support the optional external access facility.
- SDR1—The MPC855T does not support memory segments.
- Segment registers—The MPC855T does not support memory segments.

4.1.2.3 PowerPC Supervisor-Level Register Bit Assignments

This section describes bit assignments of supervisor-level PowerPC registers implemented by the MPC855T. For more details, see the *Programming Environments Manual for 32-Bit Processors*.

4.1.2.3.1 Machine State Register (MSR)

The 32-bit machine state register (MSR) is used to configure such parameters as the privilege level, whether translation is enabled, and the endian-mode. It can be read by the **mfmsr** instruction and modified by the **mtmsr**, **sc**, and **rfi** instructions.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—													POW	—	ILE
Reset	0000_0000_0000_0000															
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	—	SE	BE	—		IP	IR	DR	—		RI	LE
Reset	0	0	0	0	0	0	0	0	0	—	0	0	—		0	0
R/W	R/W															

Figure 4-3. Machine State Register (MSR)

When an exception is taken, most MSR bits are saved in the SRR1 and the MSR is reconfigured with the state of the exception handler using the values in Figure 4-3. This process is described in Section 6.1.6, “Exception Latency.”

After a hard reset, MSR[IP] takes the value specified in hard reset configuration word. See Section 11.3.1.1, “Hard Reset Configuration Word.” MSR bits are described in Table 4-8.

Table 4-8. MSR Field Descriptions

Bit(s)	Name	Description
0–12	—	Reserved
13	POW	Power management enable 0 Power management disabled (normal operation mode) 1 Power management enabled (reduced power mode) Note: Power management functions are implementation-dependent; see Section 14.5, “Power Control (Low-Power Modes).”
14	—	Reserved
15	ILE	Exception little-endian mode. When an exception occurs, this bit is copied into MSR[LE] to select the endian mode for the context established by the exception.
16	EE [1]	External interrupt enable 0 The processor delays recognition of external and decrementer interrupt conditions. 1 The processor is enabled to take an external or decrementer interrupt.
17	PR[1]	Privilege level 0 The processor can execute both user- and supervisor-level instructions. 1 The processor can only execute user-level instructions.
18	FP[1]	Floating-point available. 0 The processor prevents dispatch of floating-point instructions, including floating-point loads, stores, and moves. 1 The processor can execute floating-point instructions. (This setting is invalid on the MPC855T)

Table 4-8. MSR Field Descriptions (continued)

Bit(s)	Name	Description
19	ME[1]	Machine check enable 0 Machine check exceptions are disabled. 1 Machine check exceptions are enabled.
20	—	Reserved
21	SE[1]	Single-step trace enable (Optional) 0 The processor executes instructions normally. 1 A single-step trace exception is generated when the next instruction executes successfully. Note: If the function is not implemented, SE is treated as reserved.
22	BE[1]	Branch trace enable (Optional) 0 The processor executes branch instructions normally. 1 The processor generates a branch trace exception after completing the execution of a branch instruction, regardless of whether the branch was taken. Note: If the function is not implemented, this bit is treated as reserved.
23–24	—	Reserved
25	IP	Exception prefix. The setting of IP specifies whether an exception vector offset is prepended with Fs or 0s. In the following description, *nnnnn* is the offset of the exception vector. See Table 6-1. 0 Exceptions are vectored to the physical address 0x000*n_nnnn* 1 Exceptions are vectored to the physical address 0xFFF*n_nnnn* The reset value of IP is determined by the IIP bit (bit 2) in the hard reset configuration word. See Section 11.3.1.1, “Hard Reset Configuration Word.” Subsequent soft resets cause IP to revert to the value latched during hard reset configuration.
26	IR[1]	Instruction address translation 0 Instruction address translation is disabled. 1 Instruction address translation is enabled. For more information, see Chapter 8, “Memory Management Unit.”
27	DR[1]	Data address translation 0 Data address translation is disabled. 1 Data address translation is enabled. For more information, see Chapter 8, “Memory Management Unit.”
28–29	—	Reserved
30	RI[1]	Recoverable exception (for system reset and machine check exceptions). 0 Exception is not recoverable. 1 Exception is recoverable. For more information, see Chapter 6, “Exceptions.”
31	L E[1]	Little-endian mode enable 0 The processor runs in big-endian mode. 1 The processor runs in little-endian mode.

[1] These bits are loaded into SRR1 when an exception is taken. These bits are written back into the MSR when an **rfi** is executed.

4.1.2.3.2 Processor Version Register

The value of the PVR register’s version field is 0x0050. The value of the revision field is incremented each time the core is revised.

4.1.3 MPC855T-Specific SPRs

Table 4-2 and Table 4-9 list SPRs specific to the MPC855T. Debug registers, which have additional protection, are described in Chapter 44, “System Development and Debugging.” Supervisor-level registers are described in Table 4-9.

Table 4-9. MPC855T-Specific Supervisor-Level SPRs

SPR Number			Name	Comments	Serialize Access
Decimal	SPR[5–9]	SPR[0–4]			
80	00010	10000	EIE	See Section 6.1.5, “Recoverability after an Exception.”	Write
81	00010	10001	EID	—	Write
82	00010	10010	NRI	—	Write
631	10011	10111	DPIR [1]	—	Fetch-only
638	10011	11110	IMMR	Section 10.4.1, “Internal Memory Map Register (IMMR).”	Write (as a store)
560	10001	10000	IC_CST	Section 7.3.1, “Instruction Cache Control Registers”	Write (as a store)
561	10001	10001	IC_ADR	Section 7.3.1, “Instruction Cache Control Registers””	Write (as a store)
562	10001	10010	IC_DAT	Section 7.3.1, “Instruction Cache Control Registers””	Write (as a store)
568	10001	11000	DC_CST	Section 7.3.2, “Data Cache Control Registers”	Write (as a store)
569	10001	11001	DC_ADR	Section 7.3.2, “Data Cache Control Registers”	Write (as a store)
570	10001	11010	DC_DAT	Section 7.3.2, “Data Cache Control Registers”	Write (as a store)
784	11000	10000	MI_CTR	Section 8.8.1, “IMMU Control Register (MI_CTR)”	Write (as a store)
786	11000	10010	MI_AP	Section 8.8.10, “MMU Access Protection Registers (MI_AP/MD_AP)”	Write (as a store)
787	11000	10011	MI_EPN	Section 8.8.3, “IMMU/DMMU Effective Page Number Register (Mx_EPN)”	Write (as a store)
789	11000	10101	MI_TWC (MI_L1DL2P)	Section 8.8.4, “IMMU Tablewalk Control Register (MI_TWC)	Write (as a store)
790	11000	10110	MI_RPN	Section 8.8.6, “IMMU Real Page Number Register (MI_RPN)”	Write (as a store)
816	11001	10000	MI_CAM	Section 8.8.12.1, “IMMU CAM Entry Read Register (MI_CAM)”	Write (as a store)
817	11001	10001	MI_RAM0	Section 8.8.12.2, “IMMU RAM Entry Read Register 0 (MI_RAM0)”	Write (as a store)

Table 4-9. MPC855T-Specific Supervisor-Level SPRs (continued)

SPR Number			Name	Comments	Serialize Access
Decimal	SPR[5–9]	SPR[0–4]			
818	11001	10010	MI_RAM1	Section 8.8.13, “DMMU RAM Entry Read Register 1 (MD_RAM1)”	Write (as a store)
792	11000	11000	MD_CTR	Section 8.8.2, “DMMU Control Register (MD_CTR).”	Write (as a store)
793	11000	11001	M_CASID	Section 8.8.9, “MMU Current Address Space ID Register (M_CASID)”	Write (as a store)
794	11000	11010	MD_AP	Section 8.8.10, “MMU Access Protection Registers (MI_AP/MD_AP)”	Write (as a store)
795	11000	11011	MD_EPN	Section 8.8.3, “IMMU/DMMU Effective Page Number Register (Mx_EPN)”	Write (as a store)
796	11000	11100	M_TWB (MD_L1P)	Section 8.8.8, “MMU Tablewalk Base Register (M_TWB)”	Write (as a store)
797	11000	11101	MD_TWC (MD_L1DL2P)	Section 8.8.5, “DMMU Tablewalk Control Register (MD_TWC)”	Write (as a store)
798	11000	11110	MD_RPN	Section 8.8.7, “DMMU Real Page Number Register (MD_RPN)”	Write (as a store)
799	11000	11111	M_TW (M_SAVE)	Section 8.8.11, “MMU Tablewalk Special Register (M_TW)”	Write (as a store)
824	11001	11000	MD_CAM	Section 8.8.12.4, “DMMU CAM Entry Read Register (MD_CAM)”	Write (as a store)
825	11001	11001	MD_RAM0	Section 8.8.12.5, “DMMU RAM Entry Read Register 0 (MD_RAM0)”	Write (as a store)
826	11001	11010	MD_RAM1	Section 8.8.13, “DMMU RAM Entry Read Register 1 (MD_RAM1)”	Write (as a store)

[1] Fetch-only register; **mtspr** is ignored; using **mfspr** gives an undefined value.

Debug-level registers are described in Table 4-10. These registers are described in Section 44.5.1, “Development Support Registers.”

Table 4-10. MPC855T-Specific Debug-Level SPRs

SPR Number			Name	Serialize Access
Decimal	SPR[5–9]	SPR[0–4]		
144	00100	10000	CMPA	Fetch sync on write
145	00100	10001	CMPB	Fetch sync on write
146	00100	10010	CMPC	Fetch sync on write
147	00100	10011	CMPD	Fetch sync on write
148	00100	10100	ICR	Fetch sync on write

Table 4-10. MPC855T-Specific Debug-Level SPRs (continued)

SPR Number			Name	Serialize Access
Decimal	SPR[5–9]	SPR[0–4]		
149	00100	10101	DER	Fetch sync on write
150	00100	10110	COUNTA	Fetch sync on write
151	00100	10111	COUNTB	Fetch sync on write
152	00100	11000	CMPE	Write: Fetch sync Read: Sync relative to load/store operations
153	00100	11001	CMPF	Write: Fetch sync Read: Sync relative to load/store operations
154	00100	11010	CMPG	Write: Fetch sync Read: Sync relative to load/store operations
155	00100	11011	CMPH	Write: Fetch sync Read: Sync relative to load/store operations
156	00100	11100	LCTRL1	Write: Fetch sync Read: Sync relative to load/store operations
157	00100	11101	LCTRL2	Write: Fetch sync Read: Sync relative to load/store operations
158	00100	11110	ICTRL	Fetch sync on write
159	00100	11111	BAR	Write: Fetch sync Read: Sync relative to load/store operations. See Section 4.1.2.1, “DAR, DSISR, and BAR Operation.”
630	10011	10110	DPDR	Read and Write

4.1.3.1 Accessing SPRs

All SPRs are accessed using the **mtspr** and **mfspr** instructions, regardless of whether they are within the processor core. To access registers outside of the core, an internal bus tenure occurs using the address lines as described in Table 4-11.

Table 4-11. Addresses of SPRs Located Outside of the Core

Address Lines			
0–17	18–22	23–27	28–31
0...0	SPR[0–4]	SPR[5–9]	0000

Address errors in this tenure cause a software emulation exception.

4.2 Register Initialization at Reset

This section describes how basic registers are set under reset conditions, other register settings are described in Chapter 7, “Instruction and Data Caches,” and Chapter 8, “Memory Management Unit.”

A system reset interrupt occurs when a nonmaskable interrupt is generated either by the software watchdog timer or the assertion of $\overline{\text{IRQ0}}$. The only registers affected by the system reset interrupt are MSR, SRR0, and SRR1; no other reset activity occurs. Section 6.1.2.1, “System Reset Interrupt (0x00100),” describes values for these registers after system reset.

When a hard or soft reset occurs, registers are set in the same way, as follows:

- SRR0, SRR1—Set to an undefined value.
- MSR[IP]—Programmable through the IIP bit in the hard reset configuration word.
- MSR[ME]—Cleared.
- ICTRL—Cleared.
- LCTRL1—Cleared.
- LCTRL2—Cleared.
- COUNTA[16–31]—Cleared.
- COUNTB[16–31]—Cleared.
- ICR—Cleared (no exception occurred).
- DER[2,14,28–31]—Set (all debug-specific exceptions cause debug mode entry).

Reset values for memory-mapped registers are provided with individual register descriptions throughout this manual.

Chapter 5
MPC855T Instruction Set

This chapter describes the instructions implemented by the MPC855T. These instructions are organized by the level of architecture in which they are implemented—UISA, VEA, and OEA. These levels are described in Section 3.2.1, “Levels of the PowerPC Architecture.”

5.1 Operand Conventions

This section describes the operand conventions as they are represented in two levels of the architecture. It also provides detailed descriptions of conventions used for storing values in registers and memory, accessing the MPC855T’s registers, and representation of data in these registers.

5.1.1 Data Organization in Memory and Data Transfers

Bytes in memory are numbered consecutively starting with 0. Each number is the address of the corresponding byte.

Memory operands may be bytes, half words, words, or double words, or, for the load/store multiple and move assist instructions, a sequence of bytes or words. The address of a memory operand is the address of its first byte (that is, of its lowest-numbered byte).

5.1.2 Aligned and Misaligned Accesses

The operand of a single-register memory access instruction has a natural alignment boundary equal to the operand length. In other words, the natural address of an operand is an integral multiple of the operand length. A memory operand is said to be aligned if it is aligned at its natural boundary; otherwise it is misaligned.

Operands for single-register memory access instructions have the characteristics shown in Table 5-1 (Although not permitted as memory operands, quad words are shown because quad-word alignment is desirable for certain memory operands.).

Table 5-1. Memory Operands

Operand	Length	Addr[28–31] If Aligned
Byte	8 bits	xxxx
Half word	2 bytes	xxx0
Word	4 bytes	xx00
Double word	8 bytes	x000
Quad word	16 bytes	0000
Note: An “x” in an address bit position indicates that the bit can be 0 or 1 independent of the state of other bits in the address.		

The concept of alignment is also applied more generally to data in memory. For example, a 12-byte data item is said to be word-aligned if its address is a multiple of four.

Any memory access that crosses an alignment boundary must be broken into multiple discrete accesses. For the case of string accesses, the hardware makes no attempt to get aligned in an effort to reduce the number of discrete accesses. (Multiword accesses are architecturally required to be aligned.) The resulting performance degradation depends upon how well each individual access behaves with respect to the memory hierarchy. At a minimum, additional cache access cycles are required. More dramatically, for the case of access to a noncacheable page, each discrete access involves an individual bus operation which will reduce the effective bandwidth of the bus.

The frequent use of misaligned accesses is discouraged since they can compromise the overall performance of the processor.

5.2 Instruction Set Summary

This section describes instructions and addressing modes defined for the MPC855T. These instructions are divided into the following functional categories:

- Integer instructions—These include arithmetic and logical instructions. For more information, see Section 5.2.4.1, “Integer Instructions.”
- Load and store instructions—These include integer load and store instructions only. For more information, see Section 5.2.4.2, “Load and Store Instructions.”
- Flow control instructions—These include branching instructions, condition register logical instructions, and other instructions that affect the instruction flow. For more information, see Section 5.2.4.3, “Branch and Flow Control Instructions.”
- Trap instructions—These instructions are used to test for a specified set of conditions; see Section 5.2.4.4, “Trap Instructions,” for more information.
- Processor control instructions—These instructions are used for synchronizing memory accesses and managing caches and TLBs. For more information, see Sections 5.2.4.5, 5.2.5.1, and 5.2.6.2.

- Memory synchronization instructions—These instructions are used for memory synchronizing. See Sections 5.2.4.6 and 5.2.5.2 for more information.
- Memory control instructions—These instructions provide control of caches, and TLBs. For more information, see Sections 5.2.5.3 and 5.2.6.3.
- System linkage instructions—For more information, see Section 5.2.6.1, “System Linkage Instructions.”

Note that this grouping of instructions does not necessarily indicate the execution unit that processes a particular instruction or group of instructions. This information, which is useful in taking full advantage of the MPC855T’s parallel instruction execution, is provided in Chapter 8, “Instruction Set,” in *The Programming Environments Manual*.

Integer instructions operate on word operands. The architecture uses instructions that are four bytes long and word-aligned. It provides for byte, half word, and word operand loads and stores between memory and a set of 32 general-purpose registers (GPRs).

Arithmetic and logical instructions do not read or modify memory. To use the contents of a memory location in a computation and then modify the same or another memory location, the memory contents must be loaded into a register, modified, and then written to the target location using load and store instructions.

The description of each instruction includes the mnemonic and a formatted list of operands. To simplify assembly language programming, a set of simplified mnemonics (extended mnemonics in the architecture specification) and symbols is provided for some of the frequently-used instructions; see Appendix F, “Simplified Mnemonics,” in *The Programming Environments Manual* for a complete list of simplified mnemonic examples.

5.2.1 Classes of Instructions

The MPC855T instructions belong to one of the following three classes:

- Defined
- Illegal
- Reserved

Note that while the definitions of these terms are consistent among the MPC8xx processors, the assignment of these classifications is not. For example, an instruction that is specific to 64-bit implementations is considered defined for 64-bit implementations but illegal for 32-bit implementations such as the MPC855T.

The class is determined by examining the primary opcode and the extended opcode, if any. If the opcode, or combination of opcode and extended opcode, is not that of a defined instruction or of a reserved instruction, the instruction is illegal.

In future versions of the architecture, instruction codings that are now illegal may become assigned to instructions in the architecture, or may be reserved by being assigned to processor-specific instructions.

5.2.1.1 Definition of Boundedly Undefined

If instructions are encoded with incorrectly set bits in reserved fields, the results on execution can be said to be boundedly undefined. If a user-level program executes the incorrectly coded instruction, the resulting undefined results are bounded in that a spurious change from user to supervisor state is not allowed, and the level of privilege exercised by the program in relation to memory access and other system resources cannot be exceeded. Boundedly undefined results for a given instruction may vary between implementations, and between execution attempts in the same implementation.

5.2.1.2 Defined Instruction Class

Defined instructions are guaranteed to be supported in all MPC8xx implementations, except as stated in the instruction descriptions in Chapter 8, “Instruction Set,” in *The Programming Environments Manual*. The MPC855T provides hardware support for all instructions defined for 32-bit implementations, except floating-point instructions.

An MPC8xx processor invokes the illegal instruction error handler (part of the program exception) when the unimplemented instructions are encountered so they may be emulated in software, as required.

A defined instruction can have invalid forms, as described in the following section.

5.2.1.3 Illegal Instruction Class

Illegal instructions can be grouped into the following categories:

- Instructions that are not implemented in the architecture. These opcodes are available for future extensions of the architecture; that is, future versions of the architecture may define any of these instructions to perform new functions.

  The following primary opcodes are defined as illegal but may be used in future extensions to the architecture:

  1, 4, 5, 6, 9, 22, 56, 57, 60, 61
- Instructions that are not implemented in a specific MPC8xx implementation. For example, instructions that can be executed on 64-bit processors are considered illegal by 32-bit processors.

  The following primary opcodes are defined for 64-bit implementations only and are illegal on the MPC855T:

  2, 30, 58, 62

- All unused extended opcodes are illegal. The unused extended opcodes can be determined from information in Section A.2, “Instructions Sorted by Opcode,” in the *Programming Environments Manual* and Section 5.2.1.4, “Reserved Instruction Class.” Notice that extended opcodes for instructions that are defined only for 64-bit implementations are illegal in 32-bit implementations, and vice versa.

  The following primary opcodes have unused extended opcodes.

  17, 19, 31, 59, 63 (primary opcodes 30 and 62 are illegal for all 32-bit implementations, but as 64-bit opcodes they have some unused extended opcodes)
- An instruction consisting entirely of zeros is guaranteed to be an illegal instruction. This increases the probability that an attempt to execute data or uninitialized memory invokes the system illegal instruction error handler (a program exception). Note that if only the primary opcode consists of all zeros, the instruction is considered a reserved instruction. This is further described in Section 5.2.1.4, “Reserved Instruction Class.”

An attempt to execute an illegal instruction invokes the illegal instruction error handler (a program exception) but has no other effect. See Section 6.1.2.7, “Program Exception (0x00700),” for additional information about illegal and invalid instruction exceptions.

With the exception of the instruction consisting entirely of binary zeros, the illegal instructions are available for further additions to the architecture.

5.2.1.4 Reserved Instruction Class

Reserved instructions are allocated to specific implementation-dependent purposes not defined by the architecture. An attempt to execute an unimplemented reserved instruction invokes the illegal instruction error handler (a program exception). See Section 6.1.2.7, “Program Exception (0x00700),” for additional information about illegal and invalid instruction exceptions.

The following types of instructions are included in this class:

- Implementation-specific instructions
- Optional instructions defined by the architecture but not implemented by the MPC855T (for example, Floating Square Root (**fsqrt**) and Floating Square Root Single (**fsqrts**) instructions)

5.2.2 Addressing Modes

This section provides an overview of conventions for addressing memory and for calculating effective addresses. For more detailed information, see “Conventions,” in Chapter 4, “Addressing Modes and Instruction Set Summary,” of *Programming Environments Manual*.

5.2.2.1 Memory Addressing

A program references memory using the effective (logical) address computed by the processor when it executes a memory access or branch instruction or when it fetches the next sequential instruction.

5.2.2.2 Effective Address Calculation

An effective address (EA) is the 32-bit sum computed by the processor when executing a memory access or branch instruction or when fetching the next sequential instruction. For a memory access instruction, if the sum of the effective address and the operand length exceeds the maximum effective address, the memory operand is considered to wrap around from the maximum effective address through effective address 0, as described in the following paragraphs.

Effective address computations for both data and instruction accesses use 32-bit unsigned binary arithmetic. A carry from bit 0 is ignored.

Load and store operations have three categories of effective address generation:

- Register indirect with immediate index mode
- Register indirect with index mode
- Register indirect mode

Refer to Section 5.2.4.2.1, “Integer Load and Store Address Generation,” for further discussion of effective address generation for load and store operations.

Branch instructions have three categories of effective address generation:

- Immediate
- Link register indirect
- Count register indirect

Refer to Section 5.2.4.3.1, “Branch Instruction Address Calculation,” for further discussion of branch instruction effective address generation.

5.2.2.3 Synchronization

The synchronization described in this section refers to the state of the processor that is performing the synchronization.

5.2.2.3.1 Context Synchronization

The System Call (**sc**) and Return from Interrupt (**rfi**) instructions perform context synchronization by allowing previously issued instructions to complete before performing a change in context. Execution of one of these instructions ensures the following:

- No higher priority exception exists (**sc**).

- All previous instructions have completed to a point where they can no longer cause an exception.
- Previous instructions complete execution in the context (privilege, protection, and address translation) under which they were issued.
- The instructions following the **sc** or **rfi** instruction execute in the context established by these instructions.

5.2.2.3.2 Execution Synchronization

An instruction is execution synchronizing if all previously initiated instructions appear to have completed before the instruction is initiated or, in the case of the Synchronize (**sync**) and Instruction Synchronize (**isync**) instructions, before the instruction completes. For example, the Move to Machine State Register (**mtmsr**) instruction is execution synchronizing. It ensures that all preceding instructions have completed execution and will not cause an exception before the instruction executes, but does not ensure subsequent instructions execute in the newly established environment. For example, if the **mtmsr** sets the MSR[PR] bit, unless an **isync** immediately follows the **mtmsr** instruction, a privileged instruction could be executed or privileged access could be performed without causing an exception even though the MSR[PR] bit indicates user mode.

5.2.2.3.3 Instruction-Related Exceptions

There are two kinds of exceptions in the MPC855T—those caused directly by the execution of an instruction and those caused by an asynchronous event. Either may cause components of the system software to be invoked.

Exceptions can be caused directly by the execution of an instruction as follows:

- An attempt to execute an illegal instruction causes the illegal instruction (program exception) handler to be invoked. An attempt by a user-level program to execute the supervisor-level instructions listed below causes the privileged instruction (program exception) handler to be invoked. The MPC855T provides the following supervisor-level instructions—**dcbi**, **mfmsr**, **mfspr**, **mtmsr**, **mtspr**, **rfi**, **tlbie**, and **tlbsync**. Note that the privilege level of the **mfspr** and **mtspr** instructions depends on the SPR encoding.
- An attempt to access memory that is not available (page fault) causes the ISI exception handler to be invoked.
- An attempt to access memory with an effective address alignment that is invalid for the instruction causes the alignment exception handler to be invoked. See Section 6.1.2.6, “Alignment Exception (0x00600),” for restrictions on operand alignment.
- The execution of an **sc** instruction invokes the system call exception handler that permits a program to request the system to perform a service.

- The execution of a trap instruction invokes the program exception trap handler.

Exceptions caused by asynchronous events are described in Chapter 6, “Exceptions.”

5.2.3 Instruction Set Overview

This section provides a brief overview of the instructions implemented in the MPC855T and highlights any special information with respect to how the MPC855T implements a particular instruction. Note that the categories used in this section correspond to those used in Chapter 4, “Addressing Modes and Instruction Set Summary,” in *The Programming Environments Manual*.

Note that some of the instructions have the following optional features:

- CR Update—The dot (**.**) suffix on the mnemonic enables the update of the CR.
- Overflow option—The **o** suffix indicates that the overflow bit in the XER is enabled.

5.2.4 PowerPC UISA Instructions

The PowerPC UISA includes the base user-level instruction set (excluding a few user-level cache control, synchronization, and time base instructions), user-level registers, programming model, data types, and addressing modes. This section discusses the instructions defined in the UISA.

5.2.4.1 Integer Instructions

This section describes the integer instructions. These consist of the following:

- Integer arithmetic instructions
- Integer compare instructions
- Integer logical instructions
- Integer rotate and shift instructions

Integer instructions use the content of the GPRs as source operands and place results into GPRs, into the XER, and into condition register (CR) fields.

5.2.4.1.1 Integer Arithmetic Instructions

Table 5-2 lists the integer arithmetic instructions for the MPC855T.

Table 5-2. Integer Arithmetic Instructions

Name	Mnemonic	Syntax
Add Immediate	**addi**	**r**D,**r**A,SIMM
Add Immediate Shifted	**addis**	**r**D,**r**A,SIMM
Add	**add** (**add.addoaddo.**)	**r**D,**r**A,**r**B

Table 5-2. Integer Arithmetic Instructions (continued)

Name	Mnemonic	Syntax
Subtract From	**subf** (**subf.subfo subfo.**)	**r**D,**r**A,**r**B
Add Immediate Carrying	**addic**	**r**D,**r**A,SIMM
Add Immediate Carrying and Record	**addic.**	**r**D,**r**A,SIMM
Subtract from Immediate Carrying	**subfic**	**r**D,**r**A,SIMM
Add Carrying	**addc** (**addc.addcoaddco.**)	**r**D,**r**A,**r**B
Subtract from Carrying	**subfc** (**subfc.subfcosubfco.**)	**r**D,**r**A,**r**B
Add Extended	**adde** (**adde.addeoaddeo.**)	**r**D,**r**A,**r**B
Subtract from Extended	**subfe** (**subfe.subfeosubfeo.**)	**r**D,**r**A,**r**B
Add to Minus One Extended	**addme** (**addme.addmeoaddmeo.**)	**r**D,**r**A
Subtract from Minus One Extended	**subfme** (**subfme.subfmeosubfmeo.**)	**r**D,**r**A
Add to Zero Extended	**addze** (**addze.addzeoaddzeo.**)	**r**D,**r**A
Subtract from Zero Extended	**subfze** (**subfze.subfzeosubfzeo.**)	**r**D,**r**A
Negate	**neg** (**neg.negonego.**)	**r**D,**r**A
Multiply Low Immediate	**mulli**	**r**D,**r**A,SIMM
Multiply Low	**mullw** (**mullw.mullwomullwo.**)	**r**D,**r**A,**r**B
Multiply High Word	**mulhw** (**mulhw.**)	**r**D,**r**A,**r**B
Multiply High Word Unsigned	**mulhwu** (**mulhwu.**)	**r**D,**r**A,**r**B
Divide Word	**divw** [1] (**divw.divwodivwo.**)	**r**D,**r**A,**r**B
Divide Word Unsigned	**divwu** (**divwu.divwuodivwuo.**)	**r**D,**r**A,**r**B

1 **Implementation Note:** Attempting to use **divw** to perform either 0x80000000 ÷ -1 or <anything> ÷ 0 sets the contents of **r**D to 0x80000000 and if Rc =1, the contents CR0 are LT = 1, GT = 0, and EQ = 0. SO is set to the correct value.

Although there is no Subtract Immediate instruction, its effect can be achieved by using an **addi** instruction with the immediate operand negated. Simplified mnemonics are provided that include this negation. The **subf** instructions subtract the second operand (**r**A) from the third operand (**r**B). Simplified mnemonics are provided in which the third operand is subtracted from the second operand. See Appendix F, “Simplified Mnemonics,” in *The Programming Environments Manual* for examples.

5.2.4.1.2 Integer Compare Instructions

The integer compare instructions algebraically or logically compare the contents of **r**A with either the UIMM operand, the SIMM operand, or the contents of **r**B. The comparison is

signed for the **cmpi** and **cmp** instructions, and unsigned for the **cmpli** and **cmpl** instructions. Table 5-3 lists the integer compare instructions.

Table 5-3. Integer Compare Instructions

Name	Mnemonic	Syntax [1]
Compare Immediate	**cmpi**	**crf**D,L,**r**A,SIMM
Compare	**cmp**	**crf**D,L,**r**A,**r**B
Compare Logical Immediate	**cmpli**	**crf**D,L,**r**A,UIMM
Compare Logical	**cmpl**	**crf**D,L,**r**A,**r**B

1 **Implementation Note:** In these instructions, the L bit is applicable for 64-bit implementations. For the MPC855T, 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.

The **crf**D operand can be omitted if the result of the comparison is to be placed in CR0. Otherwise the target CR field must be specified in the instruction **crf**D field.

For more information refer to Appendix F, “Simplified Mnemonics,” in *The Programming Environments Manual*.

5.2.4.1.3 Integer Logical Instructions

The logical instructions shown in Table 5-4 perform bit-parallel operations. Logical instructions with the CR update enabled and instructions **andi.** and **andis.** set CR field CR0 to characterize the result of the logical operation. These fields are set as if the sign-extended low-order 32 bits of the result were algebraically compared to zero. Logical instructions without CR update and the remaining logical instructions do not modify the CR. Logical instructions do not affect the XER[SO], XER[OV], and XER[CA] bits.

For simplified mnemonics examples for the integer logical operations see Appendix F, “Simplified Mnemonics,” in *The Programming Environments Manual*.

Table 5-4. Integer Logical Instructions

Name	Mnemonic	Syntax
AND Immediate	**andi.**	**r**A,**r**S,UIMM
AND Immediate Shifted	**andis.**	**r**A,**r**S,UIMM
OR Immediate	**ori**	**r**A,**r**S,UIMM
OR Immediate Shifted	**oris**	**r**A,**r**S,UIMM
XOR Immediate	**xori**	**r**A,**r**S,UIMM
XOR Immediate Shifted	**xoris**	**r**A,**r**S,UIMM
AND	**and** (**and.**)	**r**A,**r**S,**r**B
OR	**or** (**or.**)	**r**A,**r**S,**r**B
XOR	**xor** (**xor.**)	**r**A,**r**S,**r**B

Table 5-4. Integer Logical Instructions (continued)

Name	Mnemonic	Syntax
NAND	**nand** (**nand.**)	**r**A,**r**S,**r**B
NOR	**nor** (**nor.**)	**r**A,**r**S,**r**B
Equivalent	**eqv** (**eqv.**)	**r**A,**r**S,**r**B
AND with Complement	**andc** (**andc.**)	**r**A,**r**S,**r**B
OR with Complement	**orc** (**orc.**)	**r**A,**r**S,**r**B
Extend Sign Byte	**extsb** (**extsb.**)	**r**A,**r**S
Extend Sign Half Word	**extsh** (**extsh.**)	**r**A,**r**S
Count Leading Zeros Word	**cntlzw** (**cntlzw.**)	**r**A,**r**S

5.2.4.1.4 Integer Rotate and Shift Instructions

Rotation operations are performed on data from a GPR, and the result, or a portion of the result, is returned to a GPR. See Appendix F, “Simplified Mnemonics,” in *The Programming Environments Manual* for a complete list of simplified mnemonics that allows simpler coding of often-used functions such as clearing the leftmost or rightmost bits of a register, left justifying or right justifying an arbitrary field, and simple rotates and shifts.

Integer rotate instructions rotate the contents of a register. The result of the rotation is either inserted into the target register under control of a mask (if a mask bit is 1 the associated bit of the rotated data is placed into the target register, and if the mask bit is 0 the associated bit in the target register is unchanged), or ANDed with a mask before being placed into the target register. The integer rotate instructions are listed in Table 5-5.

Table 5-5. Integer Rotate Instructions

Name	Mnemonic	Syntax
Rotate Left Word Immediate then AND with Mask	**rlwinm** (**rlwinm.**)	**r**A,**r**S,SH,MB,ME
Rotate Left Word then AND with Mask	**rlwnm** (**rlwnm.**)	**r**A,**r**S,**r**B,MB,ME
Rotate Left Word Immediate then Mask Insert	**rlwimi** (**rlwimi.)**	**r**A,**r**S,SH,MB,ME

The integer shift instructions perform left and right shifts. Immediate-form logical (unsigned) shift operations are obtained by specifying masks and shift values for certain rotate instructions. Simplified mnemonics are provided to make coding of such shifts simpler and easier to understand. The integer shift instructions are listed in Table 5-6.

Table 5-6. Integer Shift Instructions

Name	Mnemonic	Syntax
Shift Left Word	**slw** (**slw.**)	**r**A,**r**S,**r**B
Shift Right Word	**srw** (**srw.**)	**r**A,**r**S,**r**B

Table 5-6. Integer Shift Instructions (continued)

Name	Mnemonic	Syntax
Shift Right Algebraic Word Immediate	**srawi** (**srawi.**)	**r**A,**r**S,SH
Shift Right Algebraic Word	**sraw** (**sraw.**)	**r**A,**r**S,**r**B

5.2.4.2 Load and Store Instructions

Load and store instructions are issued and translated in program order; however, the accesses can occur out of order. Synchronizing instructions are provided to enforce strict ordering. This section describes the load and store instructions of the MPC855T, which consist of the following:

- Integer load instructions
- Integer store instructions
- Integer load and store with byte-reverse instructions
- Integer load and store multiple instructions
- Integer load and store string instructions

5.2.4.2.1 Integer Load and Store Address Generation

Integer load and store operations generate effective addresses using register indirect with immediate index mode, register indirect with index mode, or register indirect mode. See Section 5.2.2.2, “Effective Address Calculation,” for information about calculating effective addresses. Note that the MPC855T is optimized for load and store operations that are aligned on natural boundaries, and operations that are not naturally aligned may suffer performance degradation. Refer to Section 6.1.2.6.1, “Integer Alignment Exceptions,” for additional information about load and store address alignment exceptions.

5.2.4.2.2 Register Indirect Integer Load Instructions

For integer load instructions, the byte, half word, or word addressed by the EA is loaded into **r**D. Many integer load instructions have an update form, in which **r**A is updated with the generated effective address. For these forms, the EA is placed into **r**A and the memory element (byte, half word, word, or double word) addressed by EA is loaded into **r**D. Table 5-7 lists the integer load instructions.

Table 5-7. Integer Load Instructions

Name	Mnemonic	Syntax
Load Byte and Zero	**lbz**	**r**D,d(**r**A)
Load Byte and Zero Indexed	**lbzx**	**r**D,**r**A,**r**B
Load Byte and Zero with Update	**lbzu**	**r**D,d(**r**A)
Load Byte and Zero with Update Indexed	**lbzux**	**r**D,**r**A,**r**B

Table 5-7. Integer Load Instructions (continued)

Name	Mnemonic	Syntax
Load Half Word and Zero	**lhz**	**r**D**,**d(**r**A)
Load Half Word and Zero Indexed	**lhzx**	**r**D**,r**A**,r**B
Load Half Word and Zero with Update	**lhzu**	**r**D**,**d(**r**A)
Load Half Word and Zero with Update Indexed	**lhzux**	**r**D**,r**A**,r**B
Load Half Word Algebraic	**lha**	**r**D**,**d(**r**A)
Load Half Word Algebraic Indexed	**lhax**	**r**D**,r**A**,r**B
Load Half Word Algebraic with Update	**lhau**	**r**D**,**d(**r**A)
Load Half Word Algebraic with Update Indexed	**lhaux**	**r**D**,r**A**,r**B
Load Word and Zero	**lwz**	**r**D**,**d(**r**A)
Load Word and Zero Indexed	**lwzx**	**r**D**,r**A**,r**B
Load Word and Zero with Update	**lwzu**	**r**D**,**d(**r**A)
Load Word and Zero with Update Indexed	**lwzux**	**r**D**,r**A**,r**B

5.2.4.2.3 Integer Store Instructions

For integer store instructions, the contents of **r**S are stored into the byte, half word, word, or double word in memory addressed by the effective address (EA). Many store instructions have an update form, in which **r**A is updated with the EA. For these forms, the following rules apply:

- If **r**A ≠ 0, the EA is placed into **r**A.
- If **r**S = **r**A, the contents of **r**S are copied to the target memory element, then the generated EA is placed into **r**A (**r**S).

The MPC855T defines store with update instructions with **r**A = 0 and integer store instructions with the CR update option enabled (Rc[31] = 1) to be invalid forms. Table 5-8 lists integer store instructions for the MPC855T.

Table 5-8. Integer Store Instructions

Name	Mnemonic	Syntax
Store Byte	**stb**	**r**S,d(**r**A)
Store Byte Indexed	**stbx**	**r**S,**r**A,**r**B
Store Byte with Update	**stbu**	**r**S,d(**r**A)
Store Byte with Update Indexed	**stbux**	**r**S,**r**A,**r**B
Store Half Word	**sth**	**r**S,d(**r**A)
Store Half Word Indexed	**sthx**	**r**S,**r**A,**r**B
Store Half Word with Update	**sthu**	**r**S,d(**r**A)
Store Half Word with Update Indexed	**sthux**	**r**S,**r**A,**r**B
Store Word	**stw**	**r**S,d(**r**A)
Store Word Indexed	**stwx**	**r**S,**r**A,**r**B
Store Word with Update	**stwu**	**r**S,d(**r**A)
Store Word with Update Indexed	**stwux**	**r**S,**r**A,**r**B

5.2.4.2.4 Integer Load and Store with Byte-Reverse Instructions

Table 5-9 describes integer load and store with byte-reverse instructions. When used in a system operating with the default big-endian byte order, these instructions have the effect of loading and storing data in little-endian order. Likewise, when used in a system operating with little-endian byte order, these instructions have the effect of loading and storing data in big-endian order. For more information about big-endian and little-endian byte ordering, see “Byte Ordering” in Chapter 3, “Operand Conventions,” in *The Programming Environments Manual*.

Table 5-9. Integer Load and Store with Byte-Reverse Instructions

Name	Mnemonic	Syntax
Load Half Word Byte-Reverse Indexed	**lhbrx**	**r**D,**r**A,**r**B
Load Word Byte-Reverse Indexed	**lwbrx**	**r**D,**r**A,**r**B
Store Half Word Byte-Reverse Indexed	**sthbrx**	**r**S,**r**A,**r**B
Store Word Byte-Reverse Indexed	**stwbrx**	**r**S,**r**A,**r**B

5.2.4.2.5 Integer Load and Store Multiple Instructions

The integer load/store multiple instructions are used to move blocks of data to and from the GPRs. In some implementations, these instructions are 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.

When the MPC855T is operating with little-endian byte order, execution of a load or store multiple instruction causes the system alignment error handler to be invoked; see “Byte Ordering” in Chapter 3, “Operand Conventions,” in *The Programming Environments Manual* for more information. Table 5-10 lists the integer load and store multiple instructions for the MPC855T.

Table 5-10. Integer Load and Store Multiple Instructions

Name	Mnemonic	Syntax
Load Multiple Word	**lmw**	**r**D,d(**r**A)
Store Multiple Word	**stmw**	**r**S,d(**r**A)

5.2.4.2.6 Integer Load and Store String Instructions

The integer load and store string instructions allow movement of data from memory to registers or from registers to memory without concern for alignment. These instructions can be used for a short move between arbitrary memory locations or to initiate a long move between misaligned memory fields.

When the MPC855T is operating with little-endian byte order, execution of a load or store string instruction causes the system alignment error handler to be invoked; see “Byte Ordering” in Chapter 3, “Operand Conventions,” in *The Programming Environments Manual* for more information. Table 5-11 lists the integer load and store string instructions.

Table 5-11. Integer Load and Store String Instructions

Name	Mnemonic	Syntax
Load String Word Immediate	**lswi**	**r**D,**r**A,NB
Load String Word Indexed	**lswx**	**r**D,**r**A,**r**B

Table 5-11. Integer Load and Store String Instructions (continued)

Name	Mnemonic	Syntax
Store String Word Immediate	**stswi**	**r**S,**r**A,NB
Store String Word Indexed	**stswx**	**r**S,**r**A,**r**B

Load string and store string instructions may involve operands that are not word-aligned. As described in “Alignment Exception (0x00600)” in Chapter 6, “Exceptions,” in *The Programming Environments Manual*, a misaligned string operation suffers a performance penalty compared to a word-aligned operation of the same type.

When a string operation crosses a page boundary, the instruction may be interrupted by a DSI exception associated with the address translation of the second page. In this case, the MPC855T performs some or all memory references from the first page and none from the second before taking the exception. On return from the DSI exception, the load or store string instruction will re-execute from the beginning. For more information, refer to “DSI Exception (0x00300)” in Chapter 6, “Exceptions,” in *The Programming Environments Manual*.

5.2.4.3 Branch and Flow Control Instructions

Branch instructions are executed by the branch processing unit (BPU). The BPU receives branch instructions from the fetch unit and performs condition register (CR) lookahead operations on conditional branches to resolve them early, achieving the effect of a zero-cycle branch in many cases.

Some branch instructions can redirect instruction execution conditionally based on the value of bits in the CR. When the branch processor encounters one of these instructions, it scans the execution pipelines to determine whether an instruction in progress may affect the particular CR bit. If no interlock is found, the branch can be resolved immediately by checking the bit in the CR and taking the action defined for the branch instruction.

If an interlock is detected, the branch is considered unresolved and the direction of the branch is predicted using static branch prediction as described in “Conditional Branch Control” in Chapter 4, “Addressing Modes and Instruction Set Summary,” in the *Programming Environments Manual*. The interlock is monitored while instructions are fetched for the predicted branch. When the interlock is cleared, the branch processor determines whether the prediction was correct based on the value of the CR bit. If the prediction is correct, the branch is considered completed and instruction fetching continues. If the prediction is incorrect, the fetched instructions are purged, and instruction fetching continues along the alternate path. See Chapter 9, “Instruction Execution Timing” for information about how branches are executed.

5.2.4.3.1 Branch Instruction Address Calculation

Branch instructions can alter the sequence of instruction execution. Instruction addresses are always assumed to be word aligned; the processor ignores the two low-order bits of the generated branch target address.

Branch instructions compute the effective address (EA) of the next instruction address using the following addressing modes:

- Branch relative
- Branch conditional to relative address
- Branch to absolute address
- Branch conditional to absolute address
- Branch conditional to link register
- Branch conditional to count register

5.2.4.3.2 Branch Instructions

Table 5-12 lists the branch instructions. To simplify assembly language programming, a set of simplified mnemonics and symbols is provided for the most frequently used forms of branch conditional, compare, trap, rotate and shift, and certain other instructions. See Appendix F, “Simplified Mnemonics,” in *The Programming Environments Manual* for a list of simplified mnemonics.

Table 5-12. Branch Instructions

Name	Mnemonic	Syntax
Branch	**b** (**bablbla**)	target_addr
Branch Conditional	**bc** (**bcabclbcla**)	BO,BI,target_addr
Branch Conditional to Link Register	**bclr** (**bclrl**)	BO,BI
Branch Conditional to Count Register	**bcctr** (**bcctrl**)	BO,BI

5.2.4.3.3 Condition Register Logical Instructions

Condition register logical instructions, shown in Table 5-13, and the Move Condition Register Field (**mcrf**) instruction are also defined as flow control instructions.

Table 5-13. Condition Register Logical Instructions

Name	Mnemonic	Syntax
Condition Register AND	**crand**	**crb**D,**crb**A,**crb**B
Condition Register OR	**cror**	**crb**D,**crb**A,**crb**B
Condition Register XOR	**crxor**	**crb**D,**crb**A,**crb**B
Condition Register NAND	**crnand**	**crb**D,**crb**A,**crb**B
Condition Register NOR	**crnor**	**crb**D,**crb**A,**crb**B

Table 5-13. Condition Register Logical Instructions (continued)

Name	Mnemonic	Syntax
Condition Register Equivalent	**creqv**	**crb**D,**crb**A,**crb**B
Condition Register AND with Complement	**crandc**	**crb**D,**crb**A,**crb**B
Condition Register OR with Complement	**crorc**	**crb**D,**crb**A,**crb**B
Move Condition Register Field	**mcrf**	**crf**D,**crf**S

Note that if the LR update option is enabled for any of these instructions, these forms of the instructions are invalid in the MPC855T.

5.2.4.4 Trap Instructions

The trap instructions shown in Table 5-14 are provided to test for a specified set of conditions. If any of the conditions tested by a trap instruction are met, the system trap handler is invoked. If the tested conditions are not met, instruction execution continues normally.

Table 5-14. Trap Instructions

Name	Mnemonic	Syntax
Trap Word Immediate	**twi**	TO,**r**A,SIMM
Trap Word	**tw**	TO,**r**A,**r**B

See Appendix F, “Simplified Mnemonics,” in *The Programming Environments Manual* for a complete set of simplified mnemonics.

5.2.4.5 Processor Control Instructions

Processor control instructions are used to read from and write to the condition register (CR), machine state register (MSR), and special-purpose registers (SPRs), and to read from the time base register (TBU or TBL).

5.2.4.5.1 Move to/from Condition Register Instructions

Table 5-15 lists the instructions provided by the MPC855T for reading from or writing to the CR.

Table 5-15. Move to/from Condition Register Instructions

Name	Mnemonic	Syntax
Move to Condition Register Fields	**mtcrf**	CRM,**r**S
Move to Condition Register from XER	**mcrxr**	**crf**D
Move from Condition Register	**mfcr**	**r**D

5.2.4.6 Memory Synchronization Instructions—UISA

Memory synchronization instructions control the order in which memory operations are completed with respect to asynchronous events, and the order in which memory operations are seen by other processors or memory access mechanisms. See Section 7.6.6, “Atomic Memory References,” for additional information about these instructions and about related aspects of memory synchronization. Table 5-16 lists the UISA memory synchronization instructions for the MPC855T.

Table 5-16. Memory Synchronization Instructions—UISA

Name	Mnemonic	Syntax
Load Word and Reserve Indexed	**lwarx**	**r**D,**r**A,**r**B
Store Word Conditional Indexed	**stwcx.**	**r**S,**r**A,**r**B
Synchronize	**sync**	—

The **sync** instruction delays execution of subsequent instructions until previous instructions have completed to the point that they can no longer cause an exception and until all previous memory accesses are performed globally; the **sync** operation is not broadcast onto the MPC855T bus interface. Additionally, all load and store cache/bus activities initiated by prior instructions are completed. Touch load operations (**dcbt** and **dcbtst**) are required to complete at least through address translation, but not required to complete on the bus.

The functions performed by the **sync** instruction normally take a significant amount of time to complete; as a result, frequent use of this instruction may adversely affect performance. In addition, the number of cycles required to complete a **sync** instruction depends on system parameters and on the processor's state when the instruction is issued.

The proper paired use of the **lwarx** and **stwcx.** instructions allows programmers to emulate common semaphore operations such as “test and set,” “compare and swap,” “exchange memory,” and “fetch and add.” Examples of these semaphore operations can be found in Appendix E, “Synchronization Programming Examples,” in *The Programming Environments Manual*. The **lwarx** instruction must be paired with an **stwcx.** instruction with the same effective address used for both instructions of the pair. Note that the reservation granularity is 16 bytes.

The **lwarx** and **stwcx.** instructions are implemented according to the PowerPC architecture requirements. The concept behind the use of the **lwarx** and **stwcx.** instructions is that a processor may load a semaphore from memory, compute a result based on the value of the semaphore, and conditionally store it back to the same location (only if that location has not been modified since it was first read), and determine if the store was successful. The conditional store is performed based upon the existence of a reservation established by the preceding **lwarx** instruction. If the reservation exists when the store is executed, the store is performed and a bit is set in the CR. If the reservation does not exist when the store is executed, the target memory location is not modified and a bit is cleared in the CR.

If the store was successful, the sequence of instructions from the read of the semaphore to the store that updated the semaphore appear to have been executed atomically (that is, no other processor or mechanism modified the semaphore location between the read and the update), thus providing the equivalent of a real atomic operation. However, in reality, other processors may have read from the location during this operation. In the MPC855T, the reservations are made on behalf of aligned 16-byte sections of the memory address space.

The **lwarx** and **stwcx.** instructions require the EA to be aligned. Exception handling software should not attempt to emulate a misaligned **lwarx** or **stwcx.** instruction, because there is no correct way to define the address associated with the reservation.

In general, the **lwarx** and **stwcx.** instructions should be used only in system programs, which can be invoked by application programs as needed.

At most, one reservation exists simultaneously on any processor. The address associated with the reservation can be changed by a subsequent **lwarx** instruction. The conditional store is performed based upon the existence of a reservation established by the preceding **lwarx**, regardless of whether the address generated by the **lwarx** matches that generated by the **stwcx.** instruction. A reservation held by the processor is cleared by one of the following:

- Executing an **stwcx.** instruction to any address
- Attempt by another device to modify a location in the reservation granularity (16 bytes)

In write-through mode, **lwarx** and **stwcx.** do not cause a DSI exception.

The **sync** instruction guarantees that previously fetched instructions finish before any subsequent instructions are dispatched to the execution units. It does not affect fetching; instructions continue to be fetched up to the instruction queue limit, but dispatch stalls until the **sync** finishes.

The original purpose of the **sync** instruction was to synchronize coherent memory with other processors in a multiprocessor system; it makes sure that memory as seen by one processor is the same as memory seen by the other processors, and broadcasts a special signal to signal that the action is taking place. However, the MPC855T does not support this enforcement of coherency in a multiprocessor system, and it broadcasts no special synchronization signal. The MPC855T simply expects other processors not to rely on coherency of memory that it has cached in copy-back mode.

The only case where a **sync** instruction would be useful in an MPC8xx system is if software modified the page table structure associated with the SMMU only and needed to guarantee that data accesses after that instruction would be executed in the new data context. However, this is an unexpected special case; **isync** would work here, but the pipeline need not be flushed in this case, so **sync** is sufficient.

5.2.5 PowerPC VEA Instructions

The PowerPC VEA describes the semantics of the memory model that can be assumed by software processes, and includes descriptions of the cache model, cache control instructions, address aliasing, and other related issues.

5.2.5.1 Processor Control Instructions

In addition to the move to condition register instructions specified by the UISA, the VEA defines the Move from Time Base (**mftb**) instruction for reading the contents of the time base register. The **mftb** is a user-level instruction, it is shown in Table 5-17

Simplified mnemonics are provided for the **mftb** instruction so it can be coded with the TBR name as part of the mnemonic rather than requiring it to be coded as an operand. The **mftb** instruction serves as both a basic and simplified mnemonic. Assemblers recognize an **mftb** mnemonic with two operands as the basic form, and an **mftb** mnemonic with one operand as the simplified form. Simplified mnemonics are also provided for Move from Time Base Upper (**mftbu**), which is a variant of the **mftb** instruction rather than of **mfspr**. The MPC855T ignores the extended opcode differences between **mftb** and **mfspr** by ignoring bit 25 of both instructions and treating them both identically. For more information refer to Appendix F, “Simplified Mnemonics,” in *The Programming Environments Manual*.

Table 5-17. Move from Time Base Instruction

Name	Mnemonic	Syntax
Move from Time Base	**mftb**	**r**D, TBR

5.2.5.2 Memory Synchronization Instructions—VEA

Memory synchronization instructions control the order in which memory operations are completed with respect to asynchronous events, and the order in which memory operations are seen by other processors or memory access mechanisms. See Chapter 7, “Instruction and Data Caches,” for additional information about these instructions and about related aspects of memory synchronization.

Table 5-18 lists the VEA memory synchronization instructions for the MPC855T.

Table 5-18. Memory Synchronization Instructions—VEA

Name	Mnemonic	Syntax	MPC855T Notes
Enforce In-Order Execution of I/O	**eieio**	—	During execution, the LSU waits for previous accesses to terminate before beginning accesses associated with load/store instructions after an **eieio**.
Instruction Synchronize	**isync**	—	The **isync** instruction waits for all previous instructions to complete and discards any prefetched instructions, causing subsequent instructions to be refetched from memory.

5.2.5.2.1 eieio Behavior

The purpose of **eieio** is to prevent loads and stores from executing speculatively when appropriate, This might be desirable for a FIFO, where performing a read or write changes the FIFO's data. This should not be done unless it is certain that the instruction will be completed and not cancelled.

The same function as **eieio** can be accomplished by defining a memory space as having the guarded attribute in the MMU, in which case, the **eieio** instruction is redundant.

However, **eieio** could be useful in the rare event that a region where speculative accesses are not allowed lies in the middle of a non-guarded page.

5.2.5.2.2 isync Behavior

The **isync** instruction is context synchronizing, which guarantees that all of effects of previous instructions are in place and any instructions in the instruction queue are flushed (which means all instructions that were in the instruction queue need to be refetched). In the MPC855T, fetching an **isync** instruction causes fetch to stall, so that no refetching is required. On the MPC855T, writes to SPRs and MSR that effect context are automatically context synchronizing, so an **isync** is not required before these instructions. However, **isync** should be inserted after these instructions to ensure that instructions are fetched in the appropriate context. Furthermore, load/store instructions that update the MMU page tables in external memory should both be preceded and followed by an **isync**, to ensure that instructions before and after such instructions are fetched and completed in the appropriate context.

5.2.5.3 Memory Control Instructions—VEA

Memory control instructions include the following types:

- Cache management instructions
- Translation lookaside buffer (TLB) management instructions

This section describes the user-level cache management instructions defined by the VEA. See Section 5.2.6.3, “Memory Control Instructions—OEA,” for information about supervisor-level cache and translation lookaside buffer management instructions.

The instructions listed in Table 5-19 provide user-level programs the ability to manage on-chip caches.

As with other memory-related instructions, the effect of the cache management instructions on memory are weakly ordered. If the programmer needs to ensure that cache or other instructions have been performed with respect to all other processors and system mechanisms, a **sync** instruction must be placed in the program following those instructions.

Note that when data address translation is disabled (MSR[DR] = 0), the Data Cache Block Set to Zero (**dcbz**) instruction allocates a cache block in the cache and may not verify that the physical address is valid. If a cache block is created for an invalid physical address, a machine check condition may result when an attempt is made to write that cache block back to memory. The cache block could be written back as a result of the execution of an instruction that causes a cache miss and the invalid addressed cache block is the target for replacement or a Data Cache Block Store (**dcbst**) instruction.

Table 5-19 lists the cache instructions that are accessible to user-level programs.

Table 5-19. User-Level Cache Instructions

Name	Mnemonic	Syntax	MPC855T Notes
Data Cache Block Touch	**dcbt**	**r**A,**r**B	The appropriate cache block is checked for a hit. If it is a miss, the instruction is treated as a regular miss, except that bus error does not cause an exception. If no error occurs, the cache is updated.
Data Cache Block Touch for Store	**dcbtst**	**r**A,**r**B	
Data Cache Block Set to Zero	**dcbz**	**r**A,**r**B	Executes as defined in the VEA.
Data Cache Block Store	**dcbst**	**r**A,**r**B	Executes as defined in the VEA.
Data Cache Block Flush	**dcbf**	**r**A,**r**B	Executes as defined in the VEA.
Instruction Cache Block Invalidate	**icbi**	**r**A,**r**B	The MMU translates the EA and the associated instruction cache block is invalidated if hit.

5.2.6 PowerPC OEA Instructions

The PowerPC OEA includes the structure of the memory management model, supervisor-level registers, and the exception model.

5.2.6.1 System Linkage Instructions

This section describes system linkage instructions (see Table 5-20). The **sc** instruction is a user-level instruction that permits a user program to call on the system to perform a service and causes the processor to take an exception. The Return from Interrupt (**rfi**) instruction is a supervisor-level instruction that is useful for returning from an exception handler.

Table 5-20. System Linkage Instructions

Name	Mnemonic	Syntax
System Call	**sc**	—
Return from Interrupt	**rfi**	—

5.2.6.2 Processor Control Instructions—OEA

Processor control instructions are used to read from and write to the condition register (CR), machine state register (MSR), and special-purpose registers (SPRs), and to read from the time base register (TBU or TBL).

5.2.6.2.1 Move to/from Machine State Register Instructions

Table 5-15 lists the instructions provided by the MPC855T for reading from or writing to the MSR.

Table 5-21. Move to/from Machine State Register Instructions

Name	Mnemonic	Syntax
Move to Machine State Register	**mtmsr**	**r**S
Move from Machine State Register	**mfmsr**	**r**D

5.2.6.2.2 Move to/from Special-Purpose Register Instructions

Simplified mnemonics are provided for the **mtspr** and **mfspr** instructions so they can be coded with the SPR name as part of the mnemonic rather than as a numeric operand. See Appendix F, “Simplified Mnemonics,” in *The Programming Environments Manual* for simplified mnemonic examples. The **mtspr** and **mfspr** instructions are shown in Table 5-22

Table 5-22. Move to/from Special-Purpose Register Instructions

Name	Mnemonic	Syntax
Move to Special-Purpose Register	**mtspr**	SPR,**r**S
Move from Special-Purpose Register	**mfspr**	**r**D,SPR

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 encoding, with the high-order 5 bits appearing in bits 16–20 of the instruction encoding and the low-order 5 bits in bits 11–15.

If the SPR field contains a value not shown in Section 4.1, “MPC855T Register Implementation,” either the program exception handler is invoked or results are boundedly undefined.

5.2.6.3 Memory Control Instructions—OEA

This section describes memory control instructions, which include the following types:

- Cache management instructions
- TLB management instructions

Chapter 6
Exceptions

Core exceptions can be generated when an exception condition occurs. Exception sources in the MPC855T include the following:

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

Exception handling is transparent to user software and uses the same mechanism to handle all types of exceptions. When an exception is taken, control is transferred to an exception handler located at an offset defined for the type of exception encountered. The exception prefix bit, MSR[IP], determines whether this base address for the vector table resides at 0x000*n_nnnn* (IP = 0) or 0xFFF*n_nnnn* (IP = 1). Exceptions are handled in supervisor mode.

After the exception has been handled, the handler returns control to the interrupting program. As specified in the *Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture*, the core implements a precise exception model. This means that when an exception is taken, the following conditions are met:

- Subsequent instructions in the program flow are discarded.
- Previous instructions finish and write back their results.
- The address of the faulting instruction is saved in SRR0 and the machine state of the interrupted process is saved in SRR1.
- When the exception is taken, the instruction causing the exception might not have started executing, could be partially executed, or has completed, depending on the exception and instruction types. See Table 6-20.

For more information, see Section 6.1.4, "Implementing the Precise Exception Model."

6.1 Exceptions

The OEA defines a set of exceptions for processors which implement the PowerPC architecture, some of which are optional. The following sections describe exceptions implemented on the MPC855T. Those defined by the OEA are described in Section 6.1.2, “PowerPC-Defined Exceptions.” Section 6.1.3, “Implementation-Specific Exceptions,” describes implementation-specific exceptions.

All exceptions associated with memory are implemented as precise, 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.

In each exception handler, when registers SRR0 and SRR1 are saved, MSR[RI] can be set.

Table 6-1 defines the offset value by exception type and the sections that follow describe each exception in detail. Note that the base is determined by the setting of MSR[IP].

Table 6-1. Offset of First Instruction by Exception Type

Offset	Exception	Description
OEA-Defined Exceptions		
0x00000	Reserved	—
0x00100	System reset interrupt	See Section 6.1.2.1, “System Reset Interrupt (0x00100).”
0x00200	Machine check interrupt	See Section 6.1.2.2, “Machine Check Interrupt (0x00200).”
0x00300	DSI	A DSI exception is never generated by hardware, but software may branch to this location because of an data TLB error or miss exception. See Section 6.1.2.3, “DSI Exception (0x00300).”
0x00400	ISI	An ISI exception is never generated by the hardware, but software may branch to this location because of an implementation-specific instruction TLB error exception. See Section 6.1.2.4, “ISI Exception (0x00400).”
0x00500	External Interrupt	See Section 6.1.2.5, “External Interrupt Exception (0x00500).”
0x00600	Alignment	Alignment exceptions result from the following conditions: • 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 instruction is not naturally aligned when MSR[LE] = 1. • Trying to execute a multiple/string instruction when MSR[LE] = 1. See Section 6.1.2.3, “DSI Exception (0x00300).”
0x00700	Program	The MPC855T cannot generate a floating-point exception type exception. See Section 6.1.2.7, “Program Exception (0x00700).” An implementation-specific software emulation exception is generated instead of an illegal instruction type program exception. A privileged instruction program exception is generated for an on-core valid SPR field or any SPR encoded as an external SPR if SPR[0] = 1 and MSR[PR] = 1, as well as for attempts to execute supervisor-level instructions when MSR[PR] = 1. See Table 6-11.

Table 6-1. Offset of First Instruction by Exception Type (continued)

Offset	Exception	Description
0x00800	Floating-point unavailable	The MPC855T cannot generate a floating-point exception. Attempting to execute a floating-point instruction causes an implementation-specific software emulation exception (see Section 6.1.3.1, “Software Emulation Exception (0x01000)”) regardless of the setting of MSR[FP].
0x00900	Decrementer	See Section 6.1.2.8, “Decrementer Exception (0x00900).”
0x00A00–0x00B00	Reserved	—
0x00C00	System call	See Section 6.1.2.9, “System Call Exception (0x00C00).”
0x00D00	Trace	See Section 6.1.2.10, “Trace Exception (0x00D00).”
0x00E00	Floating-point assist	See Section 6.1.2.11, “Floating-Point Assist Exception.”
Implementation-Specific Exceptions		
0x01000	Software emulation	See Section 6.1.3.1, “Software Emulation Exception (0x01000).”
0x01100	Instruction TLB miss	See Section 6.1.3.2, “Instruction TLB Miss Exception (0x01100).”
0x01200	Data TLB miss	See Section 6.1.3.3, “Data TLB Miss Exception (0x01200).”
0x01300	Instruction TLB error	See Section 6.1.3.4, “Instruction TLB Error Exception (0x01300).”
0x01400	Data TLB error	See Section 6.1.3.5, “Data TLB Error Exception (0x014000).”
0x01500-0x01B00	Reserved	—
0x01C00	Data breakpoint	See Section 6.1.3.6, “Debug Exceptions (0x01C00–0x01F00).”
0x01D00	Instruction breakpoint	
0x01E00	Peripheral breakpoint	
0x01F00	Nonmaskable development port	

6.1.1 Exception Ordering

There are two types of exceptions. Instruction-related exceptions (synchronous exceptions) and asynchronous exceptions (interrupts).

Synchronous exceptions are detected while the core is processing the instruction. These exceptions are handled in strict program order and cannot be nested. A single instruction may generate multiple exceptions; however, any subsequent exceptions are not detected until the first exception is handled and control is returned to the program.

If more than one instruction in the pipeline causes an exception or if one instructions causes multiple exceptions, the first exception in program order is taken first. Subsequent instructions are flushed and additional instruction-related exceptions are handled in order.

Typically, asynchronous exceptions are generated by signals or by other hardware conditions. Table 6-2 lists the instruction-related exceptions in the order of detection within the instruction processing.

Table 6-2. Instruction-Related Exception Detection Order

Number	Exception Type	Cause
1	Trace	Trace bit asserted [1]
2	ITLB miss [2]	Instruction MMU TLB miss
3	ITLB error [2]	Instruction MMU protection/translation error
4	Machine check	Fetch error
5	Debug instruction breakpoint [2]	Match detection
6	Software emulation exception [2]	Attempt to invoke unimplemented feature
7 [3]	Privileged instruction	Attempt to execute privileged instruction in user mode
	Alignment	Load/store checking
	System call	**sc** instruction
	Trap	Trap instruction
8	DTLB miss [2]	Data TLB miss
9	DTLB error [2]	Data TLB protection/translation error
10	Machine check	Load or store access error
11	Debug L- breakpoint [2]	Match detection

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 MPC855T-specific exception.
3 Exclusive for any one instruction.

When multiple exception conditions exist, only the highest priority exception is taken, as shown in Table 6-3.

Table 6-3. Exception Priority

Priority	Exception Type	Cause
1	Development port nonmaskable interrupt	Signal from the development port
2	System reset interrupt	$\overline{\text{IRQ0}}$ assertion
3	Instruction-related exceptions	Instruction processing
4	Peripheral breakpoint request or development port maskable interrupt	Breakpoint signal from any peripheral
5	External interrupt (masked if MSR[EE] = 0)	Signal from the interrupt controller
6	Decrementer interrupt (masked if MSR[EE] = 0)	Decrementer request

6.1.2 PowerPC-Defined Exceptions

The following sections describe the exceptions as they are defined by the OEA, and describes how they are implemented on the MPC855T.

6.1.2.1 System Reset Interrupt (0x00100)

A system reset interrupt occurs when $\overline{\text{IRQ0}}$ is asserted. When the exception is taken, processing begins at offset 0x00100. A hard or soft reset also causes program execution to begin fetching at 0x00100 after the associated reset actions. Table 6-4 shows register settings.

Table 6-4. Register Settings after a System Reset Interrupt Exception

Register	Setting
SRR0	Set to the EA of the next instruction of the interrupted process.
SRR1	Saves the machine status prior to exceptions and to restore status when an **rfi** instruction is executed. 1–40 10–150 Others Loaded from MSR[16-31]. SRR1[30] is cleared only by loading a zero from MSR[RI].
MSR	IPNo change MENo change LE Value of MSR[ILE] of the interrupted process. Others0

6.1.2.2 Machine Check Interrupt (0x00200)

A machine check interrupt indication is received from the U bus in response to an address or data tenure. It is typically caused by an access for which the address does not exist or a data error occurs.

As defined in the OEA, machine check interrupts are enabled when MSR[ME] = 1. If MSR[ME] = 0 and a machine check condition is detected, the processor enters the checkstop state. The behavior of the core in checkstop state is dependent on the working mode as defined in Section 44.3.1.1, “Debug Mode Enable vs. Debug Mode Disable.” When debug mode is enabled, debug mode is entered instead of checkstop state. When debug mode is disabled, 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. See Section 11.1.3.3, “Checkstop Reset,” and Section 14.6.2, “PLL, Low-Power, and Reset Control Register (PLPRCR),” for more details. If MSR[ME] = 1, the machine check interrupt is taken. If SRR1[30] = 1, the interrupt is recoverable. Instruction fetching begins at offset 0x00200 and the registers are set as shown in Table 6-5.

Table 6-5. Register Settings after a Machine Check Interrupt Exception

Register	Setting
SRR0	Set to the EA of the instruction that caused the exception.
SRR1	1 1 for instruction fetch-related errors; 0 for load/store-related errors. 2–40 10–15 0 OthersLoaded from MSR[16-31]. SRR1[30] is cleared only by loading a zero from MSR[RI].

Table 6-5. Register Settings after a Machine Check Interrupt Exception (continued)

Register	Setting
MSR	IPNo change ME0 LE Copied from the ILE setting of the interrupted process Others0
DSISR	Set when the load/store bus is used: 0–140 15–16Set 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. 18–21Set to bits 21-24 of the instruction if X-form instruction and to bits 1-4 if D-form. 22–31Set to bits 6-15 of the instruction.
DAR	When the load/store bus is used, DAR holds the EA of the data access that caused the exception.

6.1.2.3 DSI Exception (0x00300)

DSI exceptions are never generated by the hardware. Software may branch to this location as a result of either implementation specific DTLB error interrupt or implementation specific STLB miss interrupt.

6.1.2.4 ISI Exception (0x00400)

ISI exceptions is never generated by the hardware. The software may branch to this location as a result of an implementation-specific ITLB error interrupt.

6.1.2.5 External Interrupt Exception (0x00500)

In the MPC855T the external interrupt is generated by the on-chip interrupt controller. It is software acknowledged and maskable by MSR[EE], which hardware clears automatically to disable external interrupts when any exception is taken.

When an external interrupt is detected, program execution continues until all previous instructions retire from the completion queue and the exception is assigned to the instruction last entry in the completion queue (at point B in Table 6-19). However, the following conditions must be met before the instruction at the end of the queue can retire.

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

Instructions not fitting these criteria are discarded along with any execution results. After the exception handler completes, execution resumes with the first instruction that was discarded. If all the instructions in the completion queue were allowed to complete, execution at the end of the exception handler resumes with the next instruction. External exception latency depends on the time required to reference memory. The measurement is

equal to the time taken for one of the following three events, in addition to the interval from B to E as shown in Table 6-19.

- Longest load/store multiple/string instruction used
- One bus cycle for aligned access
- Two bus cycles for unaligned access

System-level exception latency can be longer than the interval from B to E. If an instruction ahead of the exception-causing instruction also generates an exception, that exception is recognized first. If it is important to minimize exception latency, exception handlers should save the machine context and reenable exceptions as quickly as possible so pending external exceptions are handled quickly.

Register settings for the external interrupt exception are shown in Table 6-6.

Table 6-6. Register Settings after an External Interrupt

Register	Setting Description			
SRR0	Set to the effective address of the instruction that the processor would have attempted to execute next if no interrupt conditions were present.			
SRR1	0 Loaded with equivalent bits from the MSR 1–4 Cleared 5–9 Loaded with equivalent bits from the MSR 10–15 Cleared 16–31 Loaded with equivalent bits from the MSR Note that depending on the implementation, reserved bits in the MSR may not be copied to SRR1.			
MSR	POW 0 ILE — EE 0 PR 0	FP 0 ME — SE 0 BE 0	IP — IR 0 DR 0 RI 0	LE Set to value of ILE

6.1.2.6 Alignment Exception (0x00600)

This section describes conditions that can cause alignment exceptions in the processor. Similar to DSI exceptions, alignment exceptions use SRR0 and SRR1 to save the machine state and DSISR to determine the source of the exception. An alignment exception occurs when no higher priority exception exists and the implementation cannot perform a memory access for one of the following reasons:

- The operand of **lmw**, **stmw**, **lwarx**, or **stwcx.** is not aligned.
- The instruction is **lmw**, **stmw**, **lswi**, **lswx**, **stswi**, or **stswx** and the processor is in little-endian mode.
- An unaligned load or store in little-endian mode.

For **lmw**, **stmw**, **lswi**, **lswx**, **stswi**, and **stswx** instructions in little-endian mode, an alignment exception always occurs. For **lmw** and **stmw** instructions with an operand that is not aligned in big-endian mode, and for **lwarx** and **stwcx.** with an operand that is not

aligned in either endian mode, an implementation may yield boundedly-undefined results instead of causing an alignment exception. For all other cases listed above, an implementation may execute the instruction correctly instead of causing an alignment exception.

The register settings for alignment exceptions are shown in Table 6-7.

Table 6-7. Register Settings after an Alignment Exception

Register	Setting Description
SRR0	Set to the effective address of the instruction that caused the exception.
SRR1	0 Loaded with equivalent bits from the MSR 1–4 Cleared 5–9 Loaded with equivalent bits from the MSR 10–15 Cleared 16–31 Loaded with equivalent bits from the MSR Note that depending on the implementation, reserved bits in the MSR may not be copied to SRR1.
MSR	POW 0 FP 0 IP — LE Set to value of ILE ILE — ME — IR 0 EE 0 SE 0 DR 0 PR 0 BE 0 RI 0
DSISR	0–14 Cleared 15–16 For instructions that use register indirect with index addressing—set to bits 29–30 of the instruction encoding. For instructions that use register indirect with immediate index addressing—cleared 17 For instructions that use register indirect with index addressing—set to bit 25 of the instruction encoding. For instructions that use register indirect with immediate index addressing— set to bit 5 of the instruction encoding. 18–21 For instructions that use register indirect with index addressing—set to bits 21–24 of the instruction encoding. For instructions that use register indirect with immediate index addressing—set to bits 1–4 of the instruction encoding. 22–26 Set to bits 6–10 (identifying either the source or destination) of the instruction encoding. Undefined for **dcbz**. 27–31 Set to bits 11–15 of the instruction encoding (**r**A) for update-form instructions Set to either bits 11–15 of the instruction encoding or to any register number not in the range of registers loaded by a valid form instruction for **lmw**, **lswi**, and **lswx** instructions. Otherwise undefined. If there is no corresponding instruction, no alternative value can be specified.
DAR	Set to the EA of the data access as computed by the instruction causing the alignment exception.

The architecture does not support the use of a misaligned EA by load/store with reservation instructions. If one of these instructions specifies a misaligned EA, the exception handler should not emulate the instruction but should treat the occurrence as a programming error.

6.1.2.6.1 Integer Alignment Exceptions

Operations that are not naturally aligned may suffer performance degradation, depending on the processor design, the type of operation, the boundaries crossed, and the mode that the processor is in during execution. More specifically, these operations may either cause

an alignment exception or they may cause the processor to break the memory access into multiple, smaller accesses with respect to the cache and the memory subsystem.

6.1.2.7 Program Exception (0x00700)

A program exception occurs when no higher priority exception exists and one or more of the following exception conditions, which correspond to bit settings in SRR1, occur during execution of an instruction:

- An **lswx** instruction for which **r**A or **r**B is in the range of registers to be loaded (may cause results that are boundedly undefined)
- Privileged instruction—A privileged instruction type program exception is generated when the execution of a privileged instruction is attempted and the processor is operating in user mode (MSR[PR] is set). It is also generated for **mtspr** or **mfspr** instructions that have an invalid SPR field that contain one of the defined values having spr[0] = 1 and if MSR[PR] = 1. Some implementations may also generate a privileged instruction program exception if a specified SPR field (for a move to/from SPR instruction) is not defined for a particular implementation, but spr[0] = 1; in this case, the implementation may cause either a privileged instruction program exception, or an illegal instruction program exception may occur instead.
- Trap—A trap program exception is generated when any of the conditions specified in a trap instruction is met. Trap instructions are described in Section 5.2.4.4, “Trap Instructions.”

The register settings when a program exception is taken are shown in Table 6-8.

Table 6-8. Register Settings after a Program Exception

<table>
<tr><th>Register</th><th colspan="4">Setting Description</th></tr>
<tr><td>SRR0</td><td colspan="4">• Set to the EA of the instruction that causes the exception.</td></tr>
<tr><td>SRR1</td><td colspan="4">0 Loaded with equivalent bits from the MSR
1–4 Cleared
5–9 Loaded with equivalent bits from the MSR
10 Cleared
Note that only one of bits 11–14 of SRR1 can be set at a time.
11 Cleared.
12 Set for an illegal instruction program exception; otherwise cleared.
13 Set for a privileged instruction program exception; otherwise cleared.
14 Set for a trap program exception; otherwise cleared.
15 Cleared if SRR0 contains the address of the instruction causing the exception, and set if SRR0 contains the address of a subsequent instruction.
16–31 Loaded with equivalent bits from the MSR
Note that depending on the implementation, reserved bits in the MSR may not be copied to SRR1.</td></tr>
<tr><td>MSR</td><td>POW 0
ILE —
EE 0
PR 0</td><td>FP 0
ME —
SE 0
BE 0</td><td>IP —
IR 0
DR 0
RI 0</td><td>LE Set to value of ILE</td></tr>
</table>

When a program exception is taken, instruction execution resumes at offset 0x00700 from the physical base address indicated by MSR[IP].

6.1.2.8 Decrementer Exception (0x00900)

A decrementer exception occurs when no higher priority exception exists, a decrementer exception condition occurs (for example, the decrementer register has completed decremented), and MSR[EE] = 1. The decrementer register counts down, causing an exception request when it passes through zero. A decrementer exception request remains pending until the decrementer exception is taken and then it is cancelled. The decrementer implementation meets the following requirements:

- The counters for the decrementer and the time-base counter are driven by the same fundamental time base.
- Loading a GPR from the decrementer does not affect the decrementer.
- Storing a GPR value to DEC replaces the DEC contents with the value in the GPR.
- Whenever bit 0 of the decrementer changes from 0 to 1, a decrementer exception request is signaled. If multiple decrementer exception requests are received before the first can be reported, only one exception is reported.
- If the decrementer is altered by software and if bit 0 is changed from 0 to 1, an exception request is signaled.

The register settings for the decrementer exception are shown in Table 6-9.

Table 6-9. Register Settings after a Decrementer Exception

Register	Setting Description			
SRR0	Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.			
SRR1	0 Loaded with equivalent bits from the MSR 1–4 Cleared 5–9 Loaded with equivalent bits from the MSR 10–15 Cleared 16–31 Loaded with equivalent bits from the MSR Note that depending on the implementation, reserved bits in the MSR may not be copied to SRR1.			
MSR	POW 0 ILE — EE 0 PR 0	FP 0 ME — SE 0 BE 0	IP — IR 0 DR 0 RI 0	LE Set to value of ILE

When a decrementer exception is taken, instruction execution resumes at offset 0x00900 from the physical base address indicated by MSR[IP].

6.1.2.9 System Call Exception (0x00C00)

A system call exception occurs when a System Call (**sc**) instruction is executed. The effective address of the instruction following the **sc** instruction is placed into SRR0. MSR bits are saved in SRR1, as shown in Table 6-10. Then a system call exception is generated.

The system call exception causes the next instruction to be fetched from offset 0x00C00 from the physical base address indicated by the new setting of MSR[IP]. As with most other exceptions, this exception is context-synchronizing. Refer to Section 5.2.2.3.1, “Context Synchronization,” regarding actions performed by a context-synchronizing operation.

Table 6-10. Register Settings after a System Call Exception

Register	Setting Description			
SRR0	Set to the effective address of the instruction following the System Call instruction			
SRR1	0 Loaded with equivalent bits from the MSR 1–4 Cleared 5–9 Loaded with equivalent bits from the MSR 10–15 Cleared 16–31 Loaded with equivalent bits from the MSR Note that depending on the implementation, reserved bits in the MSR may not be copied to SRR1.			
MSR	POW 0 ILE — EE 0 PR 0	FP 0 ME — SE 0 BE 0	IP — IR 0 DR 0 RI 0	LE Set to value of ILE

When a system call exception is taken, instruction execution resumes at offset 0x00C00 from the physical base address indicated by MSR[IP].

6.1.2.10 Trace Exception (0x00D00)

A trace exception 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 exception does not occur after an instruction that causes an exception. The monitor/debugger software must change the vectors of other possible exception addresses to single-step these instructions. If this is unacceptable, other debug features can be used. See Chapter 44, “System Development and Debugging,” for more information. Table 6-11 shows register settings for trace exceptions.

Table 6-11. Register Settings after a Trace Exception

Register	Setting
SRR0	Set to the EA of the instruction following the executed instruction.

Table 6-11. Register Settings after a Trace Exception

Register	Setting
SRR1	1–40 10–150 Others Loaded from MSR[16-31]. SRR1[30] is cleared only by loading a zero from MSR[RI].
MSR	IP No change ME No change LE Copied from the ILE setting of the interrupted process Others 0

Execution resumes at offset 0x00D00 from the base address indicated by MSR[IP].

6.1.2.11 Floating-Point Assist Exception

The floating-point assist exception is not generated by the MPC855T. Attempting to execute a floating-point causes an instruction implementation-specific software emulation exception.

6.1.3 Implementation-Specific Exceptions

The following sections describe the MPC855T's implementation-specific exceptions.

6.1.3.1 Software Emulation Exception (0x01000)

An software emulation exception 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 exception condition).

In addition, the following registers are set:

Table 6-12. Register Settings after a Software Emulation Exception

Register	Setting
SRR0	Set to the EA of the instruction that caused the exception.
SRR1	1–40 10–150 OthersLoaded from MSR[16-31]. SRR1[30] is cleared only by loading a zero from MSR[RI].
MSR	IP No change MENo change LECopied from the ILE setting of the interrupted process Others0

Execution resumes at offset 0x01000 from the base address indicated by MSR[IP].

6.1.3.2 Instruction TLB Miss Exception (0x01100)

This type of exception occurs if MSR[IR] = 1 and an attempt is made to fetch an instruction from a page whose effective page number cannot be translated by TLB. The following registers are set:

Table 6-13. Register Settings after an Instruction TLB Miss Exception

Register	Setting
SRR0	Set to the EA of the instruction that caused the exception.
SRR1	0–30 4 1 10 1 11–150 OthersLoaded from MSR[16-31]. SRR1[30] is cleared only by loading a zero from MSR[RI].
MSR	IP No change MENo change LECopied from the ILE setting of the interrupted process Others0

Some instruction TLB registers are set to the values described in Chapter 8, “Memory Management Unit.” Execution resumes at offset 0x01100 from the base address indicated by MSR[IP].

6.1.3.3 Data TLB Miss Exception (0x01200)

This type of exception occurs when MSR[DR] = 1 and an attempt is made to access a page whose effective page number cannot be translated by TLB. The following registers are set:

Table 6-14. Register Settings after a Data TLB Miss Exception

Register	Setting
SRR0	Set to the EA of the instruction that caused the exception.
SRR1	1–40 10–150 OthersLoaded from MSR[16-31]. SRR1[30] is cleared only by loading a zero from MSR[RI].
MSR	IP No change MENo change LECopied from the ILE setting of the interrupted process Others0

Some instruction TLB registers are set to the values described in Chapter 8, “Memory Management Unit.” Execution resumes at offset 0x01200 from the base address indicated by MSR[IP].

6.1.3.4 Instruction TLB Error Exception (0x01300)

This type of exception occurs as a result of one of the following conditions if MSR[IR] = 1:

- The EA cannot be translated. Either the segment or page valid bit of this page is cleared in the translation table. Note that although the MPC855T does not implement segment registers as they are defined by the OEA, the concept of segment is retained as the memory space accessible to the level-one table descriptors.
- The fetch access violates memory protection.
- The fetch access is to guarded memory.

The following registers are set:

Table 6-15. Register Settings after an Instruction TLB Error Exception

Register	Setting
SRR0	Set to the EA of the instruction that caused the exception.
SRR1	Note that only one of bits 1, 3, and 4 will be set. 1 1 if the translation of an attempted access is not in the translation tables. Otherwise 0 2 0 3 1 if the fetch access was to guarded memory when MSR[IR] = 1. Otherwise 0 4 1 if the access is not permitted by the protection mechanism; otherwise 0. 11–150 OthersLoaded from MSR[16-31]. SRR1[30] is cleared only by loading a zero from MSR[RI].
MSR	IP No change MENo change LECopied from the ILE setting of the interrupted process Others0

Some instruction TLB registers are set to a value described in Chapter 8, “Memory Management Unit.” Execution resumes at offset 0x01300 from the base address indicated by MSR[IP].

6.1.3.5 Data TLB Error Exception (0x014000)

This type of exception occurs as a result of one of the following conditions:

- No EA 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 memory protection.
- An attempt was made to write to a page with a cleared change bit.

The following registers are set:

Table 6-16. Register Settings after a Data TLB Error Exception

Register	Setting
SRR0	Set to the EA of the instruction that caused the exception.
SRR1	1–40 10–150 OtherLoaded from MSR[16-31]. SRR1[30] is cleared only by loading a zero from MSR[RI].
MSR	IP No change MENo change LECopied from the ILE setting of the interrupted process Others0
DSISR	0 0 1 Set if the translation of an attempted access is not found in the translation tables. Otherwise, cleared 2–30 4 Set if the memory access is not permitted by the protection mechanism; otherwise cleared 5 0 6 1 for a store operation; 0 for a load operation. 7–310
DAR	Set to the EA of the data access that caused the exception.

Some instruction TLB registers are set to the values described in Chapter 8, “Memory Management Unit.” Execution resumes at offset 0x01400 from the base address indicated by MSR[IP].

6.1.3.6 Debug Exceptions (0x01C00–0x01F00)

A debug exception occurs in response to one of the following conditions:

- When there is an internal breakpoint match (for more details, see Section 44.2, “Watchpoints and Breakpoints Support”).
- When a peripheral breakpoint request is presented to the exception mechanism.
- When the development port request is presented to the exception mechanism.

The following registers are set:

Table 6-17. Register Settings after a Debug Exception

Register	Setting
SRR0	For I-breakpoints, set to the EA of the instruction that caused the exception. For L-breakpoint, set to the EA of the instruction after the one that caused the exception. For development port maskable request or a peripheral breakpoint, set to the EA of the instruction that the processor would have executed next if no exception conditions were present. If the development port request is asserted at reset, the value of SRR0 is undefined.
SRR1	1–40 10–150 OthersLoaded from MSR[16-31]. SRR1[30] is cleared only by loading a zero from MSR[RI]. If the development port request is asserted at reset, the value of SRR1 is undefined.

Table 6-17. Register Settings after a Debug Exception

Register	Setting
MSR	IP No change MENo change LECopied from the ILE setting of the interrupted process Others0
BAR	For L-bus breakpoint conditions. Set to the EA of the data access as computed by the instruction that caused the exception.
DSISR	For L-bus breakpoint conditions. Do not change.
DAR	For L-bus breakpoint conditions. Do not change.

Execution resumes from the following offsets from the base indicated by the MSR[IP]:

- 0x01D00–For an instruction breakpoint match
- 0x01C00–For a data breakpoint match
- 0x01E00–For a development port maskable request or a peripheral breakpoint
- 0x01F00–For a development port nonmaskable request

6.1.4 Implementing the Precise Exception Model

Because instructions execute in parallel they may execute out of order. To ensure that out-of-order execution does not affect data integrity, hardware ensures a precise exception model. As instructions are dispatched in-order to the execution units, they are assigned sequential positions in the six-entry completion queue, a FIFO buffer maintains program order. The completion queue is shown in Figure 3-2.

When an exception condition is encountered, previous instructions in the completion queue are allowed to complete and be retired from the completion queue. If one of these instructions generates another exception, that exception is handled first. Subsequent instructions, and any results associated with them, are flushed from the processor before instruction processing resumes at the appropriate exception vector. Before control passes to the exception handler, machine state is saved in SRR0 and SRR1.

After an exception handler executes, the machine state of the interrupted process is restored, typically by executing the **rfi** instruction, which writes bits from SRR1 to the MSR, SRR0 contains the instruction address at which fetching should resume. To correctly restore the architectural state, the CQ 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 until the store instruction reaches CQ0.

The completion queue can hold six instructions, but no more than four integer instructions. The other two instructions can be condition code or branch instructions. Long latency instructions may cause the completion queue to fill, stalling dispatch until the long latency instruction vacates the completion queue. The following instructions may cause the completion queue to fill:

- Integer divide instructions
- Instructions that affect or use resources external to the core (load/store instructions, and especially load/store string multiple/instructions)

6.1.5 Recoverability after an Exception

The processor cannot always recover from system reset and machine check interrupts, either because the conditions that cause the interrupt are catastrophic or because they caused the save/restore information in SRR0 and SRR1to be overwritten.

All other exceptions should be restartable. Registers such as SRR0 and SRR1 (and for some exceptions the data address register (DAR) and DSI status register (DSISR)) that may be affected by subsequent exceptions should be saved early in the routine to avoid being overwritten. Likewise, the saved values should be restored to those registers at the end of the handler routine in such a way that protects them from an exception before the instruction returns control to the interrupted process. Interrupts should also be masked in these areas by clearing (disabling) MSR[ME] for system reset and machine check interrupts and MSR[EE] for external interrupt, decrementer and two implementation-specific exceptions—debug port unmaskable interrupt and breakpoint interrupt in nonmaskable mode.

The recoverable exception bit (MSR[RI]) is defined to notify the exception handler code whether it is in a restartable state. The MSR[RI] shadow bit in SRR1 indicates if the exception is restartable. This bit does not need to be checked on exception types that are restartable by convention, except those previously mentioned. When an exception occurs, MSR[RI] is copied to the equivalent bit in SRR1 and cleared. When an **rfi** instruction is executed, MSR[RI] is copied from SRR1 or software can change the bit by using it the **mtmsr** instruction. The MSR[RI] bit is intended to be set by the exception handler after saving the machine state, in SRR0 and SRR1 (and DAR and DSISR if needed) and cleared by the exception handler before retrieving the machine state.

In critical code sections where MSR[EE] is cleared but SRR0 and SRR1 are not busy, MSR[RI] should remain set. In such cases, if an exception occurs, the process is restartable.

Table 6-18 lists SPRs that facilitate manipulation of MSR[RI] and MSR[EE]. Writing to these locations performs the specified operation. Attempting to read these locations is treated as an unimplemented instruction and causes a software emulation exception.

Table 6-18. Additional SPRs that Affect MSR Bits

Name	SPR	MSR[EE]	MSR[RI]	Used For
EIE	80	1	1	External interrupt enable: End of 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 exception are recoverable: End of 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 handler's epilogue

6.1.6 Exception Latency

Figure 6-1 describes significant events during exception processing.

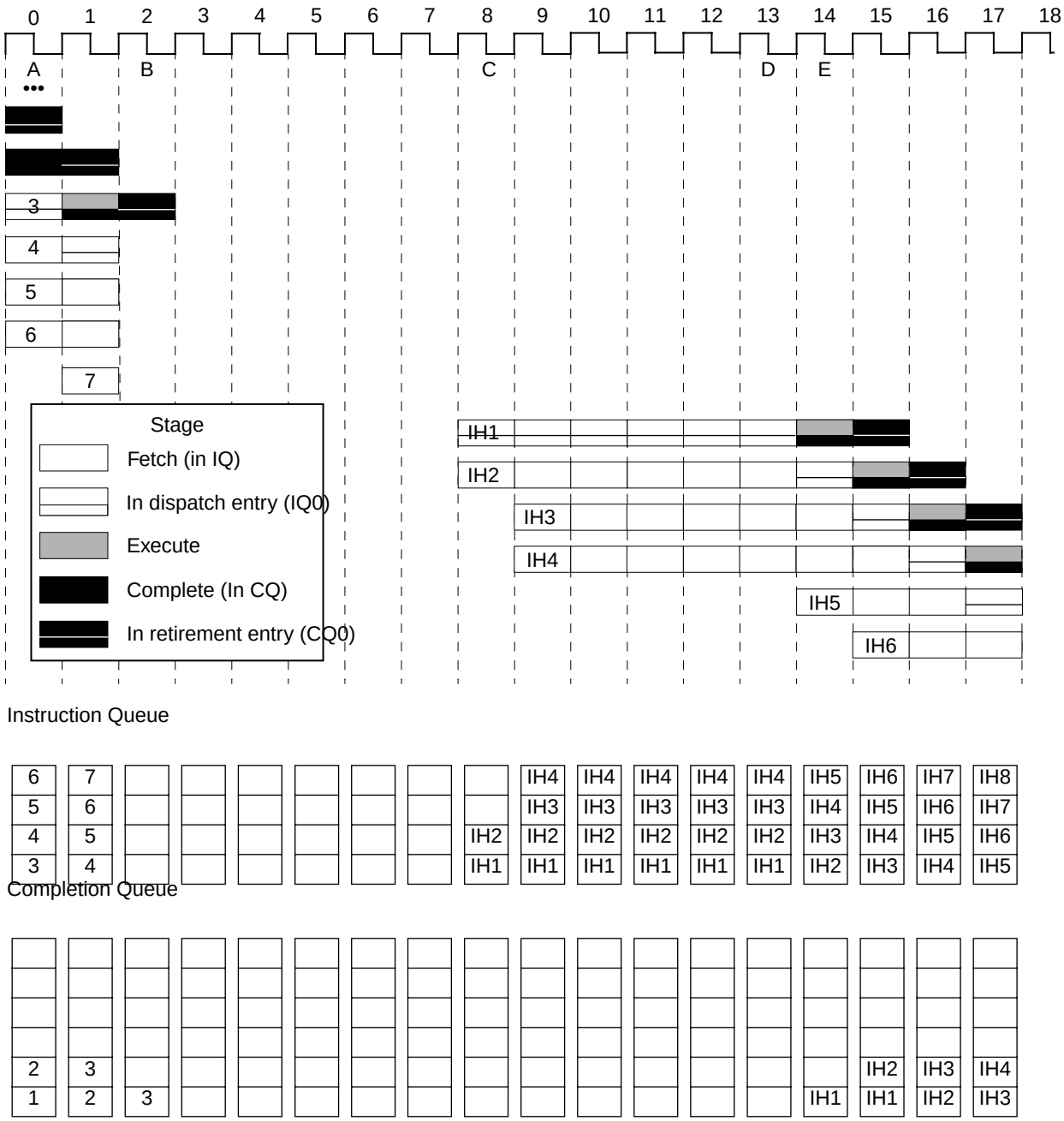


Figure 6-1. Exception Latency

Table 6-19. Exception Latency

Time Point	Fetch	Issue	Instruction Complete	Kill Pipeline
A		Faulting instruction issue		
B			Instruction complete and all previous instructions complete	

Table 6-19. Exception Latency (continued)

Time Point	Fetch	Issue	Instruction Complete	Kill Pipeline
C	Start fetch handler			Kill pipeline
D (at least 3 clocks after B)				
E		First instruction of handler dispatched		

A At time point A the excepting instruction dispatches and begins executing. Previously dispatched instructions are proceeding through the pipeline.

B The excepting instruction has executed and reached CQ0; previous instructions have finished execution without generating exceptions. The exception is recognized and between B and D (between 3 and 10 cycles) the effects of any instructions after the one that generated the interrupt are cancelled and the instructions are flushed. If the instruction had not generated an exception, it would have been retired.

C The core fetches the first instructions of the exception handler if the exception handler is external. It is 5 cycles if it is in the instruction cache and no-show mode is on.

D All state has been restored. During the interval between D and E, the machine is saving context information in the SRR0 and SRR1 registers, disabling exceptions, placing the machine in privileged mode, and fetching instructions of the exception handler. The interval between D and E requires at least one clock. The time between C and E depends on the memory system and the time it takes to fetch the first instruction of the exception handler. For full completion queue restore time, it is no less than two clocks.

E The MSR and instruction pointer of the executing process have been saved and control has been transferred to the exception handler routine. Exception handler instructions that have been fetched can be dispatched.

6.1.7 Partially Completed Instructions

Partially completed instructions can be reexecuted after the exception is handled. This precise exception model can simplify exception processing because software does not have to save the machine's internal states, unwind the pipelines, or cleanly terminate the faulting instruction stream and reverse the process to resume execution of the faulting stream.

Table 6-20. Before and After Exceptions

Exception Type	Instruction Type	Before/After	Contents of SRR0
Hard reset (caused by $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$)	Any	NA	Undefined
System reset	Any	Before	Next instruction to execute
Machine check	Any	Before	Faulting instruction
TLB miss/error [1]	Any	Before	Faulting fetch or load/store
Other noninstruction-related exceptions	Any	Before	Next instruction to execute

Table 6-20. Before and After Exceptions (continued)

Exception Type	Instruction Type	Before/After	Contents of SRR0
Alignment	Load/store	Before	Faulting instruction
Privileged instruction	Any privileged instruction	Before	Faulting instruction
Trap	**tw**, **twi**	Before	Faulting instruction
System call	sc	After	Next instruction to execute
Trace	Any	After	Next instruction to execute
Debug I- breakpoint [1]	Any	Before	Faulting instruction
Debug L- breakpoint [1]	Load/store	After	Faulting instruction + 4
Software emulation [1]	NA	Before	Faulting instruction
Floating-point unavailable	Floating-point	Before	Faulting instruction

1 Implementation-specific exceptions not defined by the PowerPC architecture

Chapter 7
Instruction and Data Caches

The MPC855T contains separate 4-Kbyte, two-way set associative instruction and data caches to allow rapid core access to instructions and data. This chapter describes the organization of the on-chip instruction and data caches, cache control, various cache operations, and the interaction between the caches, the load/store unit (LSU), the instruction sequencer, and the system interface unit (SIU).

The MPC855T cache implementation has the following characteristics:

- There are two separate 4-Kbyte instruction and data caches (Harvard architecture).
- Both instruction and data caches are two-way set associative.
- The caches implement a least-recently-used (LRU) replacement algorithm within each set.
- The cache directories are physically addressed. The physical (real) address tag is stored in the cache directory.
- Both the instruction and data caches have 16-byte cache blocks. A cache block is the block of memory that a coherency state describes, also referred to as a cache line.
- Two state bits for each data cache block allow encoding for three states:
  - Modified-valid (sometimes called ‘modified’)
  - Unmodified-valid (sometimes called ‘exclusive’)
  - Invalid
- A single state bit for each instruction cache block allows encoding for two possible states:
  - Valid
  - Invalid
- Both caches can be disabled, invalidated, or locked by issuing commands to their respective cache control registers, special-purpose registers (SPRs) specific to the MPC855T. See Section 7.3, “Cache Control Registers,” for more information.
- Individual cache blocks can be locked so that frequently accessed instructions and/or data are guaranteed to be resident in the respective cache.

On a cache miss, the MPC855T's cache blocks are filled in 16-byte bursts. The burst fill is performed as a critical-word-first operation; the critical word is simultaneously written to the cache and forwarded to the requesting unit, thus minimizing stalls due to cache fill latency. Both caches provide storage for cache tags and perform cache block replacement (LRU) function.

Both caches are tightly coupled to the MPC855T's system interface unit (SIU) to allow efficient access to the system memory controller and other bus masters. The SIU receives requests for bus operations from the instruction and data caches, and executes the operations per the external bus protocol.

The data cache provides buffers for load and store bus operations. The data cache supplies data to the GPRs by means of a 32-bit interface to the load/store unit. The LSU is directly coupled to the data cache to allow efficient movement of data to and from the general-purpose registers. The load/store unit provides all logic required to calculate effective addresses, handles data alignment to and from the data cache, and provides sequencing for load and store string and multiple operations. Write operations to the data cache can be performed on a byte, half-word, or word basis.

The instruction cache provides a 32-bit interface to the instruction sequencer. The instruction sequencer uses the instruction cache as much as possible in order to sustain the high throughput provided by the four-entry instruction queue.

7.1 Instruction Cache Organization

The MPC855T instruction cache is organized as 128 sets of two blocks, as shown in Figure 7-1. Each block consists of 16 bytes, a single state bit, a lock bit, and an address tag.

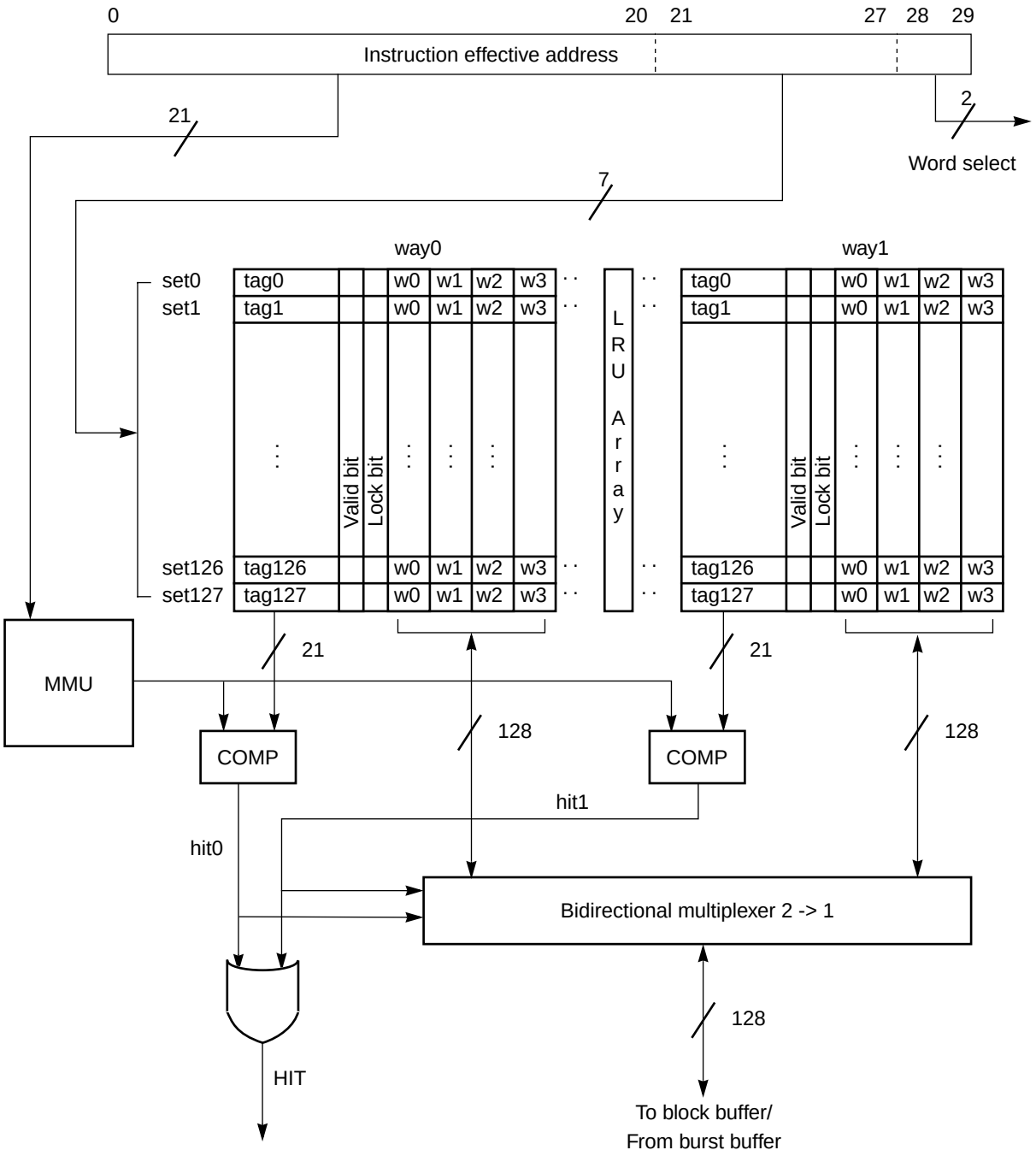


Figure 7-1. Instruction Cache Organization

Each instruction cache block contains four contiguous words from memory that are loaded from a four-word boundary; that is, bits A[28–31] of the logical (effective) addresses are zero. As a result, cache blocks are aligned with page boundaries. Also, address bits A[21–27] provide the index to select a set, and bits A[28–29] select a word within a block. The tags consist of the high-order physical address bits PA[0–20]. Address translation occurs in parallel with set selection (from A[21–27]).

The instruction cache implements a single state bit for each cache block that indicates whether the cache block is valid or invalid. The MPC855T does not support snooping of the instruction cache. All memory is considered to have memory-coherency-not-required attributes. Therefore, software must maintain instruction cache coherency. The MPC855T supports a fast instruction cache invalidate capability as described in Section 7.3.1.2.5, “Instruction Cache Invalidate All Command.”

The instruction cache also implements a lock bit for each cache block that allows instructions to be loaded into the instruction cache and locked—providing fast and deterministic execution time for critical code segments. The MPC855T supports commands for locking and unlocking individual cache blocks and for unlocking all the cache blocks at once.

7.2 Data Cache Organization

The data cache is organized as 128 sets of two blocks as shown in Figure 7-2. Each block consists of 16 bytes, two state bits, a lock bit, and an address tag.

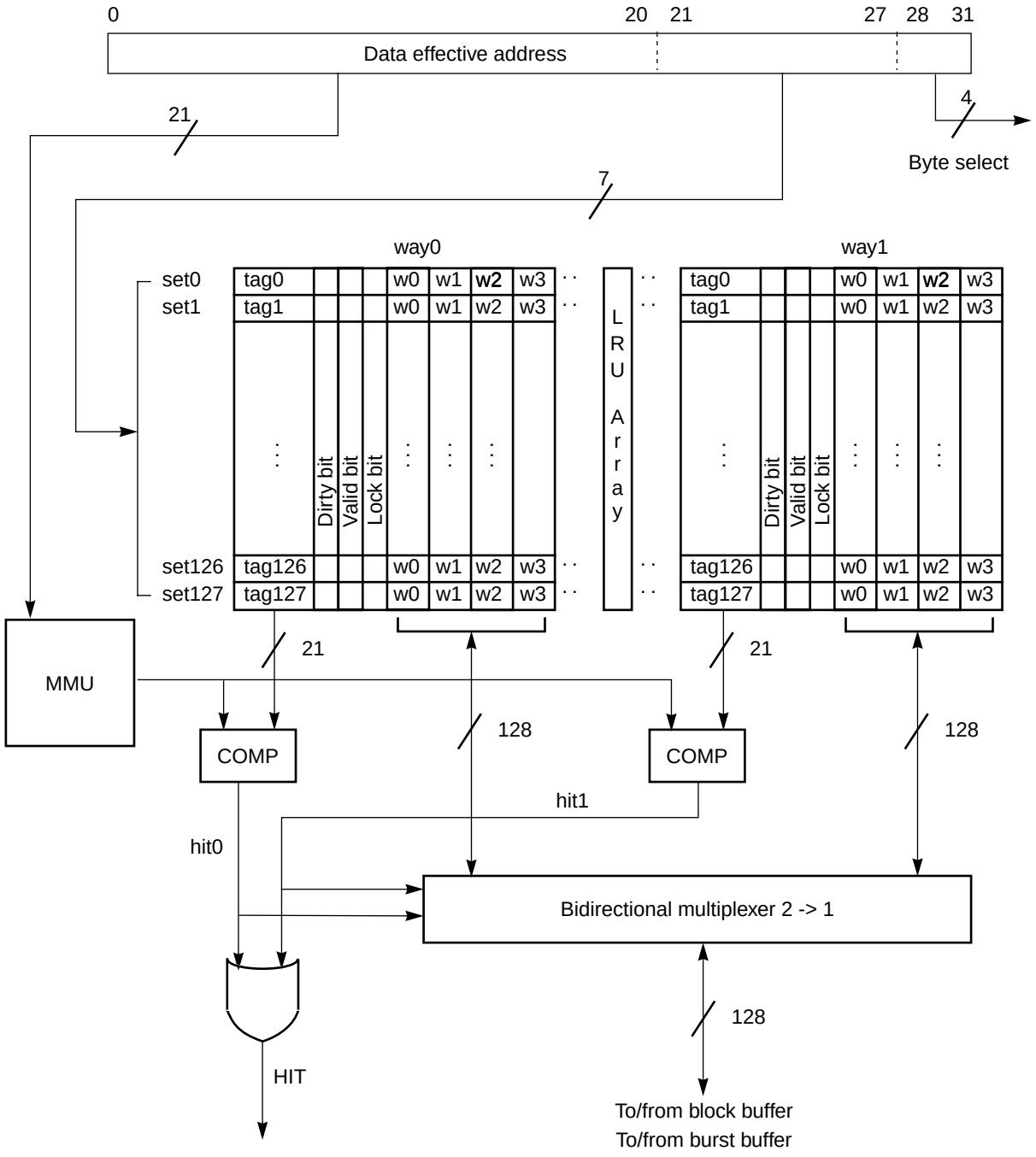


Figure 7-2. Data Cache Organization

Each cache block contains four contiguous words from memory that are loaded from a four-word boundary; that is, bits A[28–31] of the logical (effective) addresses are zero. As a result, cache blocks are aligned with page boundaries. Note that address bits A[21–27] provide the index to select a cache set. Bits A[28–31] select a byte within a block. The tags consist of the high-order physical address bits PA[0–20]. Address translation occurs in parallel with set selection (from A[21–27]).

The two state bits implement a three-state (modified-valid/unmodified-valid/invalid) protocol. The MPC855T does not provide support for snooping external bus activity. All coherency between the internal caches and external agents (memory or I/O devices) must be controlled by software.

The data cache also implements a lock bit for each cache block that allows data to be loaded into the data cache and locked. The MPC855T supports commands for locking and unlocking individual cache blocks and for unlocking all the cache blocks at once.

7.3 Cache Control Registers

The MPC855T's caches are controlled by programming commands using the cache control registers and by issuing dedicated PowerPC cache control instructions. This section describes control of the instruction and data caches by the cache control registers. Section 7.4, "PowerPC Cache Control Instructions," describes the PowerPC cache control instructions.

7.3.1 Instruction Cache Control Registers

The MPC855T implements three special purpose registers (SPRs) to control the instruction cache—the instruction cache control and status register (IC_CST), the instruction cache address register (IC_ADR), and the instruction cache data port register (IC_DAT). The instruction cache can be disabled, invalidated, or locked by issuing the appropriate commands to the instruction cache control registers (IC_CST, IC_ADR, and IC_DAT). In addition, the instruction cache control registers can be used to read the contents and tags of specific instruction cache blocks.

The **mtspr** and **mfspr** instructions are used to access the cache control registers, but they can be accessed only by supervisor-level programs (that is, when MSR[PR] = 0). Any attempt to access these SPRs with a user-level program (MSR[PR] = 1) results in a supervisor-level program exception.

The IC_CST register, shown in Figure 7-3, has an SPR encoding of 560.

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

Figure 7-3. Instruction Cache Control and Status Register (IC_CST)

Table 7-1. describes the bits of the IC_CST register.

Table 7-1. Instruction Cache Control and Status Register—IC_CST

Bits	Name	Description
0	IEN	Instruction cache enable status. 0 The instruction cache is disabled 1 The instruction cache is enabled Note that this is a read-only bit. Any attempt to write to it is ignored.
1–3	—	Reserved
4–6	CMD	Instruction cache command 000Reserved 001Cache enable 010Cache disable 011Load & lock cache block 100Unlock cache block 101Unlock all 110Invalidate all 111Reserved Note that reading these bits always returns 0b000
7–9	—	Reserved
10	CCER1	Instruction cache error type 1—bus error during an IC_CST load & load cache block command 0 No error detected 1 Error detected Note that this is a read-only, sticky bit, set only by the MPC855T when an error is detected. Reading this bit clears it.
11	CCER2	Instruction cache error type 2—no unlocked way available for an IC_CST load & lock cache block command 0 No error detected 1 Error detected Note that this is a read-only, sticky bit, set only by the MPC855T when an error is detected. Reading this bit clears it.
12–31	—	Reserved

The IC_ADR register, shown in Figure 7-4, has an SPR encoding of 561.

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	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																															

Figure 7-4. Instruction Cache Address Register (IC_ADR)

Table 7-2 describes the bits of the IC_ADR register.

Table 7-2. Instruction Cache Address Register—IC_ADR

Bits	Name	Description
0–31	ADR	Instruction cache command address. When programming the IC_CST[CMD] load & lock cache block and unlock cache block commands, IC_ADR contains the physical address in external memory of the desired cache block element. When reading the data, tags, and status contained within the instruction cache, IC_ADR is used to qualify what is to be read according to Table 7-6 See Section 7.3.1.1, “Reading Data and Tags in the Instruction Cache,” for more information.

The IC_DAT register, shown in Figure 7-5, has an SPR encoding of 562.

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	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																															

Figure 7-5. Instruction Cache Data Port Register (IC_DAT)

Table 7-3 describes the bits of the IC_DAT register.

Table 7-3. Instruction Cache Data Port Register—IC_DAT

Bits	Name	Description
0–31	DAT	Instruction cache command data. The data received when reading information from the instruction cache. See Section 7.3.1.1, “Reading Data and Tags in the Instruction Cache,” for more information.

7.3.1.1 Reading Data and Tags in the Instruction Cache

The MPC855T supports reading the data, tags, and the state and lock bits stored in the instruction cache. The instruction cache read command, issued by reading the IC_DAT register, uses the IC_ADR register to qualify what is to be read. Table 7-4. describes the fields of the IC_ADR register during an instruction cache read command.

Table 7-4. IC_ADR Fields for Cache Read Commands

0–17	18	19	20	21–27	28–29	30–31
Reserved	0 Tag 1 Data	0 Way 0 1 Way 1	Reserved	Set select (0–127)	Word select (used only for data array)	Reserved

To read the data or tags stored in the instruction cache, do the following:

1. Write the address of the data or tag to be read to the IC_ADR according to the format shown in Table 7-6
   Note that it is also possible to read this register for debugging purposes.
2. Read the IC_DAT register.

For data array (IC_ADR[18] = 1) read commands, the word selected by IC_ADR[28–29] is placed in the target general-purpose register. For tag array (IC_ADR[18] = 0) read commands, the tag and state information is placed in the target general-purpose register. Table 7-5. provides the format of the IC_DAT register for a tag read.

Table 7-5. IC_DAT Format for a Tag Read (IC_ADR[18] = 0)

0–20	21	22	23	24	25–31
Tag value	Reserved	0 Invalid 1 Valid	0 Unlocked 1 Locked	LRU bit of this set	Reserved

7.3.1.2 IC_CST Commands

All IC_CST commands, except the load & lock cache block command, are executed immediately after writing to the IC_CST register and do not generate any errors. Therefore, when executing these commands there is no need to check the error type bits in the IC_CST register. All commands should be followed by an **isync** instruction, if the instruction cache command is required to affect all instruction fetches that come after it in the program order. In addition, correct operation of the instruction cache relies on software following the procedures described in Section 7.5.5, “Updating Code and Memory Region Attributes.”

Note that when the instruction cache is executing a command, it stops handling CPU requests, which can result in machine stalls.

7.3.1.2.1 Instruction Cache Enable/Disable Commands

The instruction cache enable command (IC_CST[CMD] = 0b001) is used to enable the instruction cache; the instruction cache disable command (IC_CST[CMD] = 0b010) is used to disable the instruction cache. Neither of these commands has any error cases. The current state of the instruction cache is available by reading the instruction cache enable status bit (IC_CST[IEN]).

When disabled, the MPC855T ignores the instruction cache valid bit and operates as if all accesses have caching-inhibited access attributes (that is, all instruction fetches are propagated to the bus as single-beat transactions). Disabling the instruction cache does not affect the instruction address translation logic; MSR[IR] controls instruction address translation.

At hard reset, the instruction cache is disabled.

7.3.1.2.2 Instruction Cache Load & Lock Cache Block Command

The instruction cache load & lock cache block command (IC_CST[CMD] = 0b011) is used to lock critical code segments in the instruction cache. Locked cache blocks are not replaced during misses and are not affected by invalidate commands. Correct operation of locked instruction cache blocks relies on software following the procedures described in Section 7.5.5, “Updating Code and Memory Region Attributes.”

To load & lock one or more cache blocks:

1. Read the IC_CST error type bits to clear them.
2. Write the address of the cache block to be locked to the IC_ADR register.
3. Write the load & lock cache block command (IC_CST[CMD] = 0b011) to the IC_CST register.
4. Execute an **isync** instruction.
5. Repeat steps 2 through 4 to load & lock another cache block.
6. Read the IC_CST error type bits to determine if the sequence completed without errors.

After the load & lock cache block command is written to the IC_CST register, the cache checks if the block containing the byte addressed by IC_ADR[ADR] is in the cache (hit). If it is in the cache, the block is locked. If the block is not in the cache, a normal miss sequence is initiated (see Section 7.5.2, “Instruction Cache Miss,” for more information). After the addressed block is placed into the cache, the block is locked.

The user must check the IC_CST error type bits to determine if the load & lock cache block operation completed without error. The load & lock cache block command generates two possible errors:

- Type 1—a bus error occurred in one of the fetch cycles
- Type 2—there is no available way to lock (It is the responsibility of the user to make sure that there is at least one unlocked way in the appropriate set.)

The error type bits in the IC_CST register are sticky, thus allowing the user to perform a series of load & lock cache block commands before checking the termination status. These bits are set by the MPC855T and are cleared by software.

Note that the MPC855T considers all zero-wait-state devices on the internal bus as caching-inhibited. For this reason, software should not perform load & lock cache block

operations from these devices on the internal bus.

7.3.1.2.3 Instruction Cache Unlock Cache Block Command

The unlock cache block command (IC_CST[CMD] = 0b100) is used to unlock previously locked cache blocks. To unlock a cache block:

1. Write the address of the cache block to be unlocked to the IC_ADR register.
2. Write the unlock cache block command (IC_CST[CMD] = 0b100) to the IC_CST register.

If the block is found in the cache (hit), it is unlocked and thereafter operates as a regular valid cache block. If the block is not found in the cache (miss), no operation is performed. There are no error cases for the unlock block command.

The instruction cache performs the unlock cache block command in one clock cycle.

7.3.1.2.4 Instruction Cache Unlock All Command

The unlock all command (IC_CST[CMD] = 0b101) is used to unlock the entire instruction cache with a single command.

When the unlock all command is performed, if a cache block is locked, it is unlocked and thereafter operates as a regular valid cache block. If a block is not locked or if it is marked invalid, no operation is performed. There are no error cases for the unlock all command

The instruction cache performs the unlock all command in one clock cycle.

7.3.1.2.5 Instruction Cache Invalidate All Command

The instruction cache invalidate all command (IC_CST[CMD] = 0b110) causes all unlocked, valid blocks in the instruction cache to be marked invalid. As a result of the invalidate all command, the LRU bits of all cache blocks point to either the unlocked way or to way 0 if both ways are unlocked. There are no error cases for the invalidate all command.

The instruction cache performs the invalidate all command in one clock cycle.

7.3.2 Data Cache Control Registers

The MPC855T implements three special purpose registers (SPRs) to control the data cache—the data cache control and status register (DC_CST), the data cache address register (DC_ADR), and the data cache data port register (DC_DAT). The data cache can be disabled, invalidated, locked, or flushed by issuing the appropriate commands to the data cache control registers (DC_CST, DC_ADR, and DC_DAT). Also, the data cache control registers can be used to read the contents and tags of specific data cache blocks.

DC_CST[DFWT] can be used to force the data cache into write-through mode. DC_CST[LES] controls true-little endian byte-ordering of the MPC855T. See

Appendix A, “Byte Ordering,” for more information.

The **mtspr** and **mfspr** instructions are used to access the cache control registers, but they can be accessed only by supervisor-level programs (that is, when MSR[PR] = 0). Any attempt to access these SPRs with a user-level program (MSR[PR] = 1) results in a supervisor-level program exception.

The DC_CST register, shown in Figure 7-6, has an SPR encoding of 568.

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

Figure 7-6. Data Cache Control and Status Register (DC_CST)

Table 7-6 describes the bits of the DC_CST register.

Table 7-6. Data Cache Control and Status Register—DC_CST

Bits	Name	Description
0	DEN	Data cache enable status 0 The data cache is disabled 1 The data cache is enabled Note that this is a read-only bit. Any attempt to write to it is ignored. This bit is programmed by issuing the appropriate command in DC_CST[CMD].
1	DFWT	Data cache forced write-through 0 The write-through behavior of the data cache is determined by the write-through memory/cache access attribute (the W bit) in the MMU. 1 Writes to the data cache are forced to write through to memory. Note that this is a read-only bit. Any attempt to write to it is ignored. This bit is programmed by issuing the appropriate command in DC_CST[CMD].
2	LES	Little-endian swap 0 Used for big-endian (BE) and Modified little-endian (MOD-LE) modes. No modifications to the address or byte lanes are performed. 1 Used for true little-endian (TLE) mode. A 2-bit munge is performed on the physical address before accessing the internal U-bus. Also, for accesses originating from the PowerPC core, the SIU unmunges the address and swaps the bytes of data within each word at the external bus/internal U-bus boundary. See Appendix A, “Byte Ordering,” for more information on MPC855T byte ordering. Note that this is a read-only bit. Any attempt to write to it is ignored. This bit is programmed by issuing the appropriate command in DC_CST[CMD].
3		Reserved

Table 7-6. Data Cache Control and Status Register—DC_CST (continued)

Bits	Name	Description
4–7	CMD	Data cache command 0000 Reserved 0001 Set forced write-through bit 0010 Cache enable 0011 Clear forced write-through bit 0100 Cache disable 0101 Set true little-endian swap bit 0110 Load & lock cache block 0111 Clear little-endian swap bit 1000 Unlock cache block 1001 Reserved 1010 Unlock all 1011 Reserved 1100 Invalidate all 1101 Reserved 1110 Flush cache block 1111 Reserved Note that reading these bits always returns 0b0000
8–9		Reserved
10	CCER1	Data cache error type 1—copyback error during **dcbf** or **dcbst** instruction execution or during DC_CST flush cache block command. A machine check exception is generated when this bit is set. 0 No error detected 1 Error detected Note that this is a read-only, sticky bit, set only by the MPC855T when an error is detected. Reading this bit clears it.
11	CCER2	Data cache error type 2. This bit indicates one of two possible errors—either a bus error during DC_CST load & load cache block or flush cache block command or there is no unlocked way available for a DC_CST load & lock cache block or flush cache block command. 0 No error detected 1 Error detected Note that this is a read-only, sticky bit, set only by the MPC855T when an error is detected. Reading this bit clears it.
12–31	—	Reserved

The DC_ADR register, shown in Figure 7-7., has an SPR encoding of 569.

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	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																															

Figure 7-7. Data Cache Address Register (DC_ADR)

Table 7-7 describes the bits of the DC_ADR register.

Table 7-7. Data Cache Address Register—DC_ADR

Bits	Name	Description
0–31	ADR	Data cache command address. When programming the DC_CST load & lock cache block, unlock cache block, and flush cache block commands, DC_ADR contains the physical address of the desired cache block element in external memory. When reading the data, tags, and status contained within the data cache, DC_ADR is used to qualify what is to be read according to Table 7-7. See Section 7.3.2.1, “Reading Data Cache Tags and Copyback Buffer,” for more information.

The DC_DAT register, shown in Figure 7-8, has an SPR encoding of 570.

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

Figure 7-8. Data Cache Data Port Register (DC_DAT)

Table 7-8 describes the bits of the DC_DAT register.

Table 7-8. Data Cache Data Port Register—DC_DAT

Bits	Name	Description
0–31	DAT	Data cache command data. The data received when reading information from the data cache. See Section 7.3.2.1, “Reading Data Cache Tags and Copyback Buffer,” for more information.

7.3.2.1 Reading Data Cache Tags and Copyback Buffer

The MPC855T supports reading the tags, the state bits and the lock bits stored in the data cache as well as the last copyback address, and data words in the copyback buffer. The data cache read command, issued by reading DC_DAT, uses the DC_ADR register to qualify what is to be read. Table 7-9. describes the fields of the DC_ADR register during a data cache read command.

Table 7-9. DC_ADR Fields for Cache Read Commands

0–17	18	19	20	21–27	28–31
Reserved	0 Tags	0 Way 0 1 Way 1	Reserved	Set select (0–127)	Reserved
	1 Copyback buffer	Reserved		Copyback buffer address/ data-word select	

To read the copyback buffer data or the tags stored in the data cache, do the following:

1. Write the address of the copyback buffer or tag to be read to the DC_ADR according to the format shown in Table 7-9..
   Note that it is also possible to read this register for debugging purposes.
2. Read the DC_DAT register. Note that writing to the DC_DAT register is illegal. A write to DC_DAT results in an undefined data cache state.

For tag array (DC_ADR[18] = 0) read commands, the tag and state information is placed in the target general-purpose register. Table 7-10. provides the format of the DC_DAT register for a tag read.

Table 7-10. DC_DAT Format for a Tag Read (DC_ADR[18] = 0)

0–20	21	22	23	24	25	26–31
Tag value	Reserved	0 Invalid 1 Valid	0 Unlocked 1 Locked	LRU bit of this set	0 Unmodified 1 Modified	Reserved

The last copyback address or data buffer can be read by using the copyback buffer read command (DC_ADR[18] = 1). The copyback buffer select field (DC_ADR[21–27]), shown in Table 7-11, determines which word of the cache block in the copyback buffer is read.

Table 7-11. Copyback Buffer Select Field (DC_ADR[21–27]) Encoding

DC_ADR[21–27]	Buffer Selected
0x00	Copyback buffer data word 0
0x01	Copyback buffer data word 1
0x02	Copyback buffer data word 2
0x03	Copyback buffer data word 3
0x04	Copyback address

7.3.2.2 DC_CST Commands

All DC_CST commands, except the load & lock cache block and flush cache block commands, are executed immediately after writing to the DC_CST register and do not generate any errors. Therefore, there is no need to check the error type bits in the DC_CST register except when executing the load & lock cache block and flush cache block commands.

Note that when the data cache is executing a command, it stops handling CPU requests, which can result in machine stalls.

7.3.2.2.1 Data Cache Enable/Disable Commands

The data cache enable command (DC_CST[CMD] = 0b0010) is used to enable the data cache; the data cache disable command (DC_CST[CMD] = 0b0100) is used to disable the data cache. Neither of these commands has any error cases. The current state of the data

cache is available by reading the data cache enable status bit (DC_CST[DEN]).

When disabled, the MPC855T ignores the data cache state bits and operates as if all accesses have caching-inhibited access attributes (that is, all accesses are propagated to the bus as single-beat transactions). Disabling the data cache does not affect the data address translation logic; MSR[DR] controls data address translation.

Note that the data cache is disabled at hard reset. Also, the data cache is automatically disabled when a type 1 data cache error (see Table 7-6 for DC_CST[CCER1] conditions) generates a machine check exception.

7.3.2.2.2 Data Cache Load & Lock Cache Block Command

The data cache load & lock cache block command (DC_CST[CMD] = 0b0110) is used to lock critical data in the data cache. Locked cache blocks are not replaced during misses and are not affected by invalidate commands.

To load & lock one or more cache blocks:

1. Read the DC_CST error type bits to clear them.
2. Write the address of the cache block to be locked to the DC_ADR register.
3. Write the load & lock cache block command (DC_CST[CMD] = 0b0110) to the DC_CST register.
4. Repeat steps 2 and 3 to load & lock another cache block.
5. Read DC_CST[CCER2] to determine if the sequence completed without errors.

After the load & lock cache block command is written to the DC_CST register, the cache checks if the block containing the byte addressed by DC_ADR[ADR] is in the cache (hit). If it is in the cache, the block is locked and the command terminates with no exception. If the block is not in the cache, a normal miss sequence is initiated (see Section 7.6, “Data Cache Operation,” for more information). After the addressed block is placed into the cache, the block is locked.

The user must check DC_CST[CCER2] to determine if the load & lock cache block operation completed without error. The error type bits in the DC_CST register are sticky, thus allowing the user to perform a series of load & lock commands before checking the termination status. These bits are set by the MPC855T and are cleared by software.

Note that the MPC855T considers all zero-wait-state devices on the internal bus as caching-inhibited. For this reason, software should not perform load & lock operations from these devices on the internal bus.

7.3.2.2.3 Data Cache Unlock Cache Block Command

The unlock cache block command (DC_CST[CMD] = 0b1000) is used to unlock previously locked cache blocks. To unlock a cache block:

1. Write the address of the cache block to be unlocked to the DC_ADR register.

2. Write the unlock cache block command (DC_CST[CMD] = 0b1000) to the DC_CST register.

If the block is found in the cache (hit), it is unlocked and thereafter operates as a regular valid cache block. If the block is not found in the cache (miss), no operation is performed. There are no error cases for the unlock block command.

The data cache performs the unlock cache block command in one clock cycle.

7.3.2.2.4 Data Cache Unlock All Command

The data cache unlock all command (DC_CST[CMD] = 0b1010) is used to unlock the entire data cache with a single command. When the unlock all command is performed, if a cache block is locked, it is unlocked and thereafter operates as a regular valid cache block. If a block is not locked or if it is marked invalid, no operation is performed. There are no error cases for the unlock all command.

The data cache performs the unlock all command in one clock cycle.

7.3.2.2.5 Data Cache Invalidate All Command

The data cache invalidate all command (DC_CST[CMD] = 0b1100) causes all unlocked, valid blocks in the data cache to be marked invalid, regardless of whether the data is modified. Therefore, this command may effectively destroy modified data. To invalidate the entire data cache the invalidate all command should be preceded by an unlock all command. Note that the data cache is not automatically invalidated at hard reset.

As a result of the invalidate all command, the LRU bits of all cache blocks point to either the unlocked way or to way 0 if both ways are unlocked. There are no error cases for the invalidate all command.

The data cache performs the invalidate all command in one clock cycle.

7.3.2.2.6 Data Cache Flush Cache Block Command

The data cache flush cache block command (DC_CST[CMD] = 0b1110) is used to write the contents of an unlocked, modified-valid cache block to memory and subsequently invalidate that cache block. If the cache block is unmodified-valid, the cache block is invalidated without writing the contents to memory. If the cache block is locked or if it is marked invalid, no operation is performed.

If a bus error occurs while executing the DC_CST flush cache block command, DC_CST[CCER1] is set and a machine check exception is generated. The data of the cache block flagged by the bus error is contained in the copyback buffer; it will have already been flushed from the data cache array. See Section 7.3.2.1, “Reading Data Cache Tags and Copyback Buffer,” for more information.

The PowerPC cache control instructions **dcbst** and **dcbf** can also be used to flush the data cache. Note that the PowerPC cache control instructions operate on effective addresses that are translated while the DC_CST flush cache block command operates on a physically addressed block contained within the data cache. When there is a need to restrict the flushing to a specific memory area or to maintain architectural compliance, it is

recommended to use the PowerPC cache control instructions; when there is a need to flush the entire data cache and there is no concern for architectural compliance, using the DC_CST flush cache block command is more efficient.

7.4 PowerPC Cache Control Instructions

The PowerPC architecture defines instructions for controlling both the instruction and data caches. The cache control instructions, **icbi, dcbt**, **dcbtst**, **dcbz**, **dcbst**, **dcbf**, and **dcbi**, are intended for the management of the local caches. In the following descriptions, the memory/cache access attributes refer to the write-through/write-back, caching-inhibited/caching-allowed, guarded/not guarded status of the addressed page.

Note that the MPC855T does not broadcast cache control instructions nor does it snoop such broadcasts.

A TLB miss exception is generated if the effective address of one of these instructions cannot be translated and data address relocation is enabled. A TLB error exception is generated if these instructions encounter a TLB protection violation.

7.4.1 Instruction Cache Block Invalidate (icbi)

The effective address is computed, translated, and checked for protection violations as defined in the PowerPC architecture. This instruction is treated as a store with respect to address translation and memory protection. If the address hits an unlocked block in the instruction cache, the cache block is placed in the invalid state. If the address misses in the instruction cache or if the block is locked, no action is taken. The function of this instruction is independent of the memory/cache access attributes.

This command is not privileged and has no associated error cases. The instruction cache performs the **icbi** instruction in one clock cycle. To accurately calculate the latency of this instruction, bus latency should be taken into consideration.

7.4.2 Data Cache Block Touch (dcbt) and Data Cache Block Touch for Store (dcbtst)

The Data Cache Block Touch (**dcbt**) and Data Cache Block Touch for Store (**dcbtst**) instructions provide potential system performance improvement through the use of software-initiated prefetch hints. The MPC855T treats these instructions identically (that is, a **dcbtst** instruction behaves exactly the same as a **dcbt** instruction on the MPC855T).

The MPC855T loads the data into the cache when the effective address hits in the TLB, is permitted load access from the addressed page, and is directed at a caching-allowed page. Otherwise, the MPC855T treats these instructions as no-ops. The data brought into the cache as a result of this instruction is validated in the same manner that a load instruction would be (that is, it is marked as unmodified-valid). Note that the successful execution of

the **dcbt** (or **dcbtst**) instruction affects the state of the TLB and cache LRU bits.

7.4.3 Data Cache Block Zero (dcbz)

The effective address is computed, translated, and checked for protection violations as defined in the PowerPC architecture. The **dcbz** instruction is treated as a store to the addressed byte with respect to address translation and protection.

If the block containing the byte addressed by the EA is in the data cache, all bytes are cleared, and the tag is marked as modified-valid. If the block containing the byte addressed by the EA is not in the data cache and the corresponding page is caching-allowed, the block is established in the data cache without fetching the block from main memory, and all bytes of the block are cleared, and the tag is marked as modified-valid.

The **dcbz** instruction executes regardless of whether the cache block is locked, but if the cache is disabled, an alignment exception is generated. If the page containing the byte addressed by the EA is caching-inhibited or write-through, then the system alignment exception handler is invoked.

7.4.4 Data Cache Block Store (dcbst)

The effective address is computed, translated, and checked for protection violations as defined in the PowerPC architecture. This instruction is treated as a load with respect to address translation and memory protection.

If the address hits in the cache and the cache block is in the unmodified-valid state, no action is taken. If the address hits in the cache and the cache block is in the modified-valid state, the modified block is written back to memory and the cache block is placed in the unmodified-valid state.

If a bus error occurs while executing the **dcbst** instruction, DC_CST[CCER1] is set and a machine check exception is generated. The data of the cache block flagged by the bus error is retrieved from the copyback buffer, not from the data cache. See Section 7.3.2.1, “Reading Data Cache Tags and Copyback Buffer,” for more information.

The function of this instruction is independent of the memory/cache access attributes. The **dcbst** instruction executes regardless of whether the cache is disabled or the cache block is locked.

7.4.5 Data Cache Block Flush (dcbf)

The effective address is computed, translated, and checked for protection violations as defined in the PowerPC architecture. This instruction is treated as a load with respect to address translation and memory protection.

If the address hits in the cache, and the block is in the modified-valid state, the modified

block is written back to memory and the cache block is placed in the invalid state. If the address hits in the cache, and the cache block is in the unmodified-valid state, the cache block is placed in the invalid state. If the address misses in the cache, no action is taken.

If a bus error occurs while executing the **dcbf** instruction, DC_CST[CCER1] is set and a machine check exception is generated. The data of the cache block flagged by the bus error is retrieved from the copyback buffer, not from the data cache. See Section 7.3.2.1, “Reading Data Cache Tags and Copyback Buffer,” for more information.

The function of this instruction is independent of the memory/cache access attributes. The **dcbf** instruction executes regardless of whether the cache is disabled or the cache block is locked.

7.4.6 Data Cache Block Invalidate (dcbi)

The effective address is computed, translated, and checked for protection violations as defined in the PowerPC architecture. This instruction is treated as a store with respect to address translation and memory protection.

If the address hits in the cache, the cache block is placed in the invalid state, regardless of whether the data is modified. If the address misses in the cache, no action is taken. Because this instruction may effectively destroy modified data, it is privileged (that is, **dcbi** is available only to programs at the supervisor privilege level, MSR[PR] = 0).

The function of this instruction is independent of the memory/cache access attributes. The **dcbi** instruction executes regardless of whether the cache is disabled or the cache block is locked.

7.5 Instruction Cache Operations

When the instruction MMU is enabled (MSR[IR] = 1), the instruction cache operates as defined by the memory/cache access attributes. When the instruction MMU is disabled (MSR[IR] = 0), the instruction cache operates as defined by the default instruction memory access attributes. The default state of the caching-inhibited/caching-allowed attribute is determined by MI_CTR[CIDEF], and the entire memory space defaults to the guarded attribute. See Chapter 8, “Memory Management Unit,” for more information.

An instruction cache access begins with an instruction fetch request from the instruction sequencer in the PowerPC core. As shown in Figure 7-1., bits 21–27 of the instruction address provide the index to select a set (0–127) within the instruction cache array. The tags from each way of the set are compared against bits 0–20 of the instruction address. If a match is found and the matched entry is valid, then it is a cache hit. If no tag matches or the matched tag is not valid, it is a cache miss.

The data path for the instruction cache and its surrounding logic are shown in Figure 7-9.

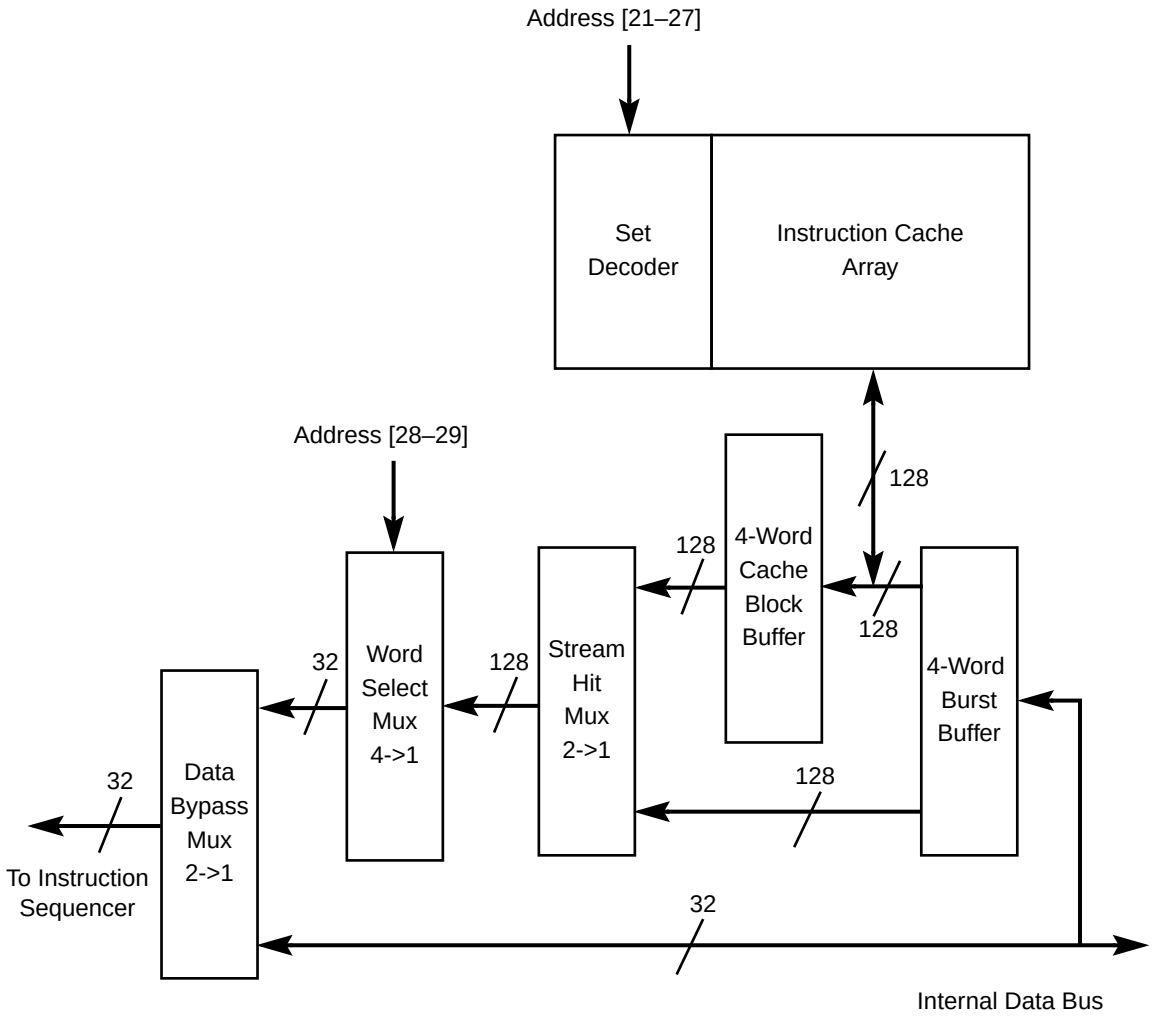


Figure 7-9. Instruction Cache Data Path

The 4-word burst buffer holds the last cache block received from the internal bus (the last miss); the 4-word block buffer holds the last block retrieved from the instruction cache (the last hit). Note that if one of these buffers contains the requested instruction, it is also considered a cache hit. To minimize power consumption, the MPC855T can detect that one of the buffers contains the requested instruction and service the instruction request from the buffers without activating the instruction cache array.

The MPC855T instruction cache includes the following operational features:

- Instruction fetch latency is reduced by sending the requested instruction address to the instruction cache and internal bus simultaneously. A cache hit aborts the internal bus transaction before the MPC855T can initiate an external fetch.
- The instruction cache supports stream hits (allows fetching from the burst buffer or directly from the internal data bus, before the instruction cache array is filled)

- The instruction cache supports hits under misses (allows servicing hits while a previous miss is being fetched from the external bus)
- A fetch request from the instruction sequencer has priority over burst buffer writes to the cache array (the burst buffer holds the last missed cache block), thus increasing the overall performance
- Efficiently uses the pipeblock of the internal data bus by initiating a new burst cycle (if miss is detected) while bringing the tail of the previous missed block
- Performance for caching-inhibited regions is enhanced by fetching a full 4-word block into the burst buffer. Instructions in the burst buffer are only used once before being refetched

7.5.1 Instruction Cache Hit

In the case of a cache hit, the cache block is transferred to the cache block buffer and forwarded to the stream hit multiplexer and word select multiplexer. As shown in Figure 7-2, bits 28–29 of the instruction address are used to select one word of the cache block which is transferred to the instruction sequencer in the core.

7.5.2 Instruction Cache Miss

On an instruction cache miss, the address of the missed instruction is driven on the internal bus with a 4-word burst transfer read request. The transfer begins with the word requested by the instruction sequencer (critical-word first), followed by the remaining words (if any) of the cache block, then by any remaining words at the beginning of the block (wrap-around).

On a cache miss, the critical word is simultaneously written to the burst buffer and forwarded to the instruction sequencer, thus minimizing stalls due to cache fill latency. As subsequent words are received from the internal bus, they are also written into the burst buffer and delivered to the instruction sequencer either directly from the internal bus or from the burst buffer (a stream hit). A cache block in the array is then selected to receive the cache block being gathered in the burst buffer. The selection algorithm gives first priority to invalid blocks. If all blocks in the set are marked invalid, the block in way 0 is selected. If none of the blocks in the selected set are invalid, then the least recently used block is selected for replacement. Locked cache blocks are never replaced.

The instruction cache is not blocked to internal accesses while the fetch (caused by a cache miss) completes. This functionality is sometimes referred to as ‘hits under misses,’ because the cache can service a hit while a cache miss fill is waiting to complete. If no bus errors are encountered during the 4-word cache block fetch, the burst buffer is marked valid and written to the cache array, provided the cache array is not busy servicing a hit.

If a bus error is encountered while fetching the requested instruction (the critical word), then a machine check exception is generated. If a bus error occurs while fetching subsequent words in the cache block, then the burst buffer is marked invalid and the cache

block is not written to the cache array.

7.5.3 Instruction Fetching on a Predicted Path

The core implements branch prediction to allow branches to issue as early as possible. This mechanism allows instruction prefetching to continue while an unresolved branch is being computed and the condition is being evaluated. Instructions fetched after unresolved branches are said to be 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 MPC855T instruction cache does not initiate a miss sequence in most cases when the instruction is inside a predicted path. The MPC855T instruction cache evaluates fetch requests to see if they are inside a predicted path. If a hit is detected, the requested instruction is delivered to the core. However, if it is a cache miss, the miss sequence is not initiated in most cases until the core finishes the branch evaluation.

7.5.4 Fetching Instructions from Caching-Inhibited Regions

The caching-inhibited/caching-allowed attributes of a memory region are programmed in the memory management unit (MMU). To improve performance when fetching instructions from caching-inhibited regions, the MPC855T loads the burst buffer with a full 4-word block. Instructions that are stored in the burst buffer and originate from a cache-inhibited region, can be sent to the instruction sequencer, at most, once before being refetched.

If an instruction fetch from a caching-inhibited region results in a cache hit, the instruction is delivered to the instruction sequencer in the core from the cache and not from memory. However, it is considered a programming error if an instruction fetch from a caching-inhibited region results in a cache hit. Software must ensure that instructions from a caching-inhibited region have not been previously loaded into the cache, or, if so, those blocks have been flushed from the cache. See Section 7.5.5, “Updating Code and Memory Region Attributes,” for more information.

It is also considered a programming error to perform load & lock cache block operations from zero wait state devices that are located on the internal bus. The MPC855T considers these devices as caching-inhibited memory regions. If a load & lock cache block operation is performed from such a device, the instruction is not guaranteed to be fetched from the instruction cache; in most cases, the instruction is fetched from the device, regardless of whether it is in the instruction cache.

7.5.5 Updating Code and Memory Region Attributes

The instruction cache does not perform snooping, so if a processor modifies a memory location that may be contained in the instruction cache, software must ensure that such memory updates are visible to the instruction fetching mechanism. Also, whenever the memory/cache attributes of any memory region are changed, it is critical that the cache contents reflect the new attributes. Therefore, when updating code or changing memory

region attributes (in the MMU) the user must perform the following steps:

1. Update code/change memory region attributes
2. Execute a **sync** instruction to ensure the update/change operation finished
3. Unlock all locked cache blocks containing code that was updated
4. Invalidate all cache blocks containing code that was updated
5. Execute an **isync** instruction

7.6 Data Cache Operation

When the data MMU is enabled (MSR[DR] = 1), the data cache operates as defined by the memory/cache access attributes. When the data MMU is disabled (MSR[DR] = 0), the data cache operates as defined by the default data memory access attributes. The default state of the write-through/write-back attribute is determined by MD_CTR[WTDEF]; the caching-inhibited/caching-allowed attribute is determined by MD_CTR[CIDEF]; and the entire memory space defaults to the guarded attribute. See Chapter 8, “Memory Management Unit,” for more information.

A data cache access begins with a load or store request from the load/store unit (LSU) in the core. The data cache has a 32-bit data path to and from the load/store unit, allowing for a 4-byte transfer per cycle. As shown in Figure 7-2., bits 21–27 of the data address provide the index to select a set (0–127) within the data cache array. The tags from both ways of the set are compared against bits 0–20 of the data address. If a match is found and the matched entry is valid, then it is a cache hit. If neither tag matches or the matched tag is not valid, it is a cache miss.

The data cache operates in both write-through and write-back modes as programmed by the memory/cache access attributes. These modes affect store hit and store miss behavior of the data cache. Load hits and load misses behave the same regardless of the write-through/write-back mode. If two logical blocks map to the same physical block, it is considered a programming error for them to specify different cache write policies.

Each data cache block contains two state bits that implement a three-state (modified-valid/unmodified-valid/invalid) protocol. The MPC855T does not support snooping of the data cache. All memory is considered to have memory coherency not required attributes. Therefore, software must maintain data cache coherency. The MPC855T does not provide support for snooping external bus activity. All coherency between the internal caches and external agents (memory or I/O devices) must be controlled by software. In addition, there is no mechanism provided for DMA or other internal masters to access the data cache directly.

The MPC855T data cache includes the following operational features:

- Single-cycle cache access on hit and one clock latency added for miss
- The data cache supports hits under load misses
- 1-word store buffer

- Store misses bypass the data cache (no-allocate store miss) in write-through mode
- 4-word copyback buffer holds replaced modified cache blocks until they can be written to memory
- Cache operation is blocked until the cache block is written to the cache array for store misses in write-back mode,
- The data cache supports the **sync** instruction through a cache pipe clean indication to the core.

7.6.1 Data Cache Load Hit

In the case of a data cache load hit, the requested word is transferred to the load/store unit. The LRU state of the set is updated, but the state bits remain unchanged.The access time for a data cache load hit is one clock cycle (that is, zero wait states).

7.6.2 Data Cache Read Miss

In the case of a data cache load miss, a block in the cache array is selected to receive the data from memory. The selection algorithm gives first priority to invalid blocks. If both blocks in the set are marked invalid, the block in way 0 is selected. If neither of the two blocks in the selected set are invalid, then the least recently used block is selected for replacement. If the replacement block is marked modified-valid, then it is temporarily stored in a copyback buffer to be written to memory later. Locked cache blocks are never replaced.

After a cache block has been selected, the word-aligned physical address of the requested data is sent to the SIU with a 4-word burst transfer read request. The SIU arbitrates for the bus and initiates the burst read. The transfer begins with the aligned word containing the requested data (critical word first), followed by the remaining words of the cache block (if any), then by any remaining words at the beginning of the block (wrap-around).

The critical word is simultaneously written to the burst buffer and forwarded to the load/store unit, thus minimizing stalls due to cache fill latency. The data cache is not blocked to internal accesses while the load (caused by a cache miss) completes. This functionality is sometimes referred to as ‘hits under misses,’ because the cache can service a hit while a cache miss fill is waiting to complete. If no bus errors are encountered during the 4-word cache block load, the burst buffer is written to the cache array (provided the cache array is not busy servicing a hit) and the cache block is marked unmodified-valid.

If a bus error is encountered while loading the requested data (the critical word), then a machine check exception is generated. If a bus error occurs while loading subsequent words in the cache block, then the cache block is marked invalid.

After the cache block with the requested data has been loaded from memory, the modified-valid cache block in the copyback buffer is sent to the SIU to be written to

memory. If a bus error is encountered during the copyback, a machine check exception is generated (the copyback error is an imprecise exception). The address and data in the copyback buffer can be read as specified in Section 7.3.2.1, “Reading Data Cache Tags and Copyback Buffer.”

7.6.3 Write-Through Mode

In write-through mode, store operations always update memory. The write-through mode is used when external memory and internal cache images must always agree. Write-through mode provides a lower worst case exception latency at the expense of average performance (for example, if it does not have to perform flush accesses).

7.6.3.1 Data Cache Store Hit in Write-Through Mode

In the case of a data cache store hit in write-through mode, the data is written into both the cache block and to memory. The LRU state of the set is updated, but the state bits remain unchanged. If a bus error is encountered during the write operation to memory, the cache block is still updated, but a machine check exception is generated.

7.6.3.2 Data Cache Store Miss in Write-Through Mode

In the case of a store miss in write-through mode, the data is only written to memory, not to the data cache. This is sometimes referred to as a‘no-allocate’ store miss because the data cache does not allocate a cache block in the cache array for the missed store operation. The state and LRU bits remain unchanged. If a bus error is encountered during the write operation to memory, a machine check exception is generated.

7.6.4 Write-Back Mode

In write-back mode, store operations do not necessarily update external memory. Data is only copied to external memory when a copyback operation is required (or the cache is deliberately flushed). For this reason the write-back mode is the preferred mode of operation when it is necessary to minimize external bus utilization and as a side effect, reduce operational power consumption.

7.6.4.1 Data Cache Store Hit in Write-Back Mode

In the case of a data cache store hit in write-back mode, the cache operation depends on the state bits of the cache block. If the store hit is to a modified-valid cache block, then data is stored in the cache block and the block stays marked modified-valid. If the store hit is to a unmodified-valid cache block, then data is stored in the cache block and the block is marked modified-valid. In either case, the LRU state of the set is updated to reflect the hit.

7.6.4.2 Data Cache Store Miss in Write-Back Mode

In the case of a data cache store miss in write-back mode, the data cache must establish the block in the cache array before modifying that block. Therefore, a block in the cache array is selected to receive the data from memory and from the load/store unit. The selection

algorithm gives first priority to invalid blocks. If both blocks in the set are marked invalid, the block in way 0 is selected. If neither of the two blocks in the selected set are invalid, then the least recently used block is selected for replacement. If the replacement block is marked modified-valid, then it is temporarily stored in the copyback buffer to be written to memory later. Locked cache blocks are never replaced.

After a cache block has been selected, the word-aligned physical address of the store data is sent to the SIU with a 4-word burst transfer read request. The SIU arbitrates for the bus and initiates a burst read. The transfer begins with the aligned word containing the requested data (critical word first), followed by the remaining words of the cache block (if any), then by any remaining words at the beginning of the block (wrap-around). As the critical word is received from the internal bus, it is merged in the burst buffer with the store data from the load/store unit. If no bus errors are encountered during the burst buffer fill operation, the cache block is written into the cache array and marked modified-valid. The data cache does not support further requests until the entire block is written to the cache array. If the machine has stalled waiting for the store to complete, execution is allowed to resume when the cache block is written into the cache array.

If a bus error is encountered while loading the target data cache block, even on a word not accessed by the load/store unit, then the cache block is not modified, and a machine check exception is generated.

After the cache block with the requested data has been loaded from memory, the cache block in the copyback buffer is sent to the SIU to be written to memory. The data cache can support further requests, as long as they hit in the cache, while performing the copyback to memory. If a bus error is encountered during the copyback, a machine check exception is generated (the copyback error is an imprecise exception). The address and data in the copyback buffer can be read as specified in Section 7.3.2.1, “Reading Data Cache Tags and Copyback Buffer.”

7.6.5 Data Accesses to Caching-Inhibited Memory Regions

For load misses to caching-inhibited memory regions, the data is read from memory but not placed in the cache and the cache status is not affected.

For store misses to caching-inhibited memory regions, the data is written to memory but not placed in the cache and the cache status is not affected.

It is considered a programming error if a load, store, or **dcbz** targeting a caching-inhibited memory region results in a cache hit. The PowerPC architecture allows the result of such programming errors to be boundedly undefined. Software must ensure that data from a caching-inhibited regions have not been previously loaded into the data cache, or, if they have, that those blocks have been flushed from the cache. Whenever the memory/cache attributes of any memory region are changed (for example, from caching-allowed to caching-inhibited), it is critical that the cache contents reflect the new attributes. Therefore,

when changing memory region attributes (in the MMU) the user must perform the procedures described in Section 7.5.5, “Updating Code and Memory Region Attributes.”.

7.6.6 Atomic Memory References

The PowerPC architecture defines the Load Word and Reserve Indexed (**lwarx**) and the Store Word Conditional Indexed (**stwcx.**) instructions to provide an atomic update function for a single, aligned word of memory. These instructions can be used to develop a rich set of multiprocessor synchronization primitives. For detailed information on these instructions, refer to Section 5.2.4.6, “Memory Synchronization Instructions—UISA,” in this book and Chapter 8, “Instruction Set,” in *The Programming Environments Manual*.

The **lwarx** instruction performs a load word from memory operation and creates a reservation for the 16-byte section of memory that contains the accessed word. The reservation granularity is 16 bytes. The **lwarx** instruction makes a nonspecific reservation with respect to the executing processor and a specific reservation with respect to other masters. This means that any subsequent **stwcx.** executed by the same processor, regardless of address, will cancel the reservation. Also, any bus write operation from another processor to an address that matches the reservation address will cancel the reservation.

The **stwcx.** instruction does not check the reservation for a matching address. The **stwcx.** instruction is only required to determine whether a reservation exists. The **stwcx.** instruction performs a store word operation only if the reservation exists. If the reservation has been cancelled for any reason, then the **stwcx.** instruction fails and clears the CR0[EQ] bit in the condition register. The architectural intent is to follow the **lwarx**/**stwcx.** instruction pair with a conditional branch which checks to see whether the **stwcx.** instruction failed.

Note that atomic memory references constructed using **lwarx**/**stwcx.** instructions depend on the presence of a coherent memory system for correct operation. These instructions should not be expected to provide atomic access to noncoherent memory. Since the MPC855T does not snoop external bus activity, provision is made to cancel a reservation inside the MPC855T by using the $\overline{\text{CR}}$ and $\overline{\text{KR}}$ input signals. The state of the reservation is always presented onto the $\overline{\text{RSV}}$ output signal. This can be used by external agents to determine when an internal condition has caused a change in the reservation state. See Section 13.4.9, “Memory Reservation,” for more information. Internal to the MPC855T, the data cache has snoop logic to monitor the internal bus for communication processor module (CPM) accesses of the address associated with the last **lwarx** instruction.

If a memory region is marked caching-allowed, the MPC855T assumes that it is the single master in the system to that region. If a caching-allowed **lwarx** or **stwcx.** access misses in the data cache, the transaction on the internal and external buses do not have a reservation. If the memory region is marked caching-inhibited or the cache is locked, and the access misses, then the **lwarx** instruction appears on the bus as a single-beat load with the reservation.

lwarx and **stwcx.** accesses to write-through memory regions do not generate DSI exceptions. The MPC855T's data cache treats all **stwcx.** operations as write-through independent of the memory/cache access attributes. When the write-through operation completes successfully on the external bus, then the data cache entry is updated (assuming it hits), and CR0[EQ] is modified to reflect the success of the operation. If the reservation is not intact, the **stwcx.** cancels the external bus transaction, and the cache block is not altered.

7.7 Cache Initialization after Reset

At power-on and hard reset, both caches are disabled. Although disabled, the cache state is preserved to enable the user to investigate the exact state of the cache prior to the event that caused the reset. To ensure proper operation after reset, initialize the instruction cache by performing the following:

1. Write the unlock all command (IC_CST[CMD] = 0b101) to the IC_CST register
2. Write the invalidate all command (IC_CST[CMD] = 0b110) to the IC_CST register
3. Write the cache enable command (IC_CST[CMD] = 0b001) to the IC_CST register

Similarly, to ensure proper operation after reset, initialize the data cache by performing the following:

1. Write the unlock all command (DC_CST[CMD] = 0b1010) to the DC_CST register
2. Write the invalidate all command (DC_CST[CMD] = 0b1100) to the DC_CST register
3. Write the cache enable command (DC_CST[CMD] = 0b0010) to the DC_CST register

After the caches are initialized, all the cache blocks are invalidated, and the LRU bits point to way 0 of each set.

7.8 Debug Support

The MPC855T can be debugged either in debug mode or by a software monitor debugger. In both cases the core of the MPC855T asserts the internal freeze signal. See Chapter 44, "System Development and Debugging," for a detailed description of the MPC855T debug support.

7.8.1 Instruction and Data Cache Operation in Debug Mode

The development system interface of the MPC855T uses the development port, which is a dedicated serial port. The development port is a relatively inexpensive interface that allows a development system to operate in a lower frequency than the core's frequency and controls system activity when the core is in debug mode. See Section 44.3, "Development

System Interface," for more information.

When the MPC855T is in debug mode, all instructions are fetched from the development port, regardless of the address generated by the MPC855T core. Therefore, the instruction cache is bypassed when the MPC855T is in debug mode. In addition, the data cache is frozen in debug mode. Loads and stores in debug mode always target system memory, regardless of whether the accessed data is resident in the data cache. The only way to access the contents of the instruction or data cache in debug mode is by using the IC_DAT or DC_DAT registers.

7.8.2 Instruction and Data Cache Operation with a Software Monitor Debugger

With debug mode disabled, a software monitor debugger can use the development support registers to assert the internal freeze signal during run-time. See Section 44.4, "Software Monitor Debugger Support," for more information.

When the internal freeze signal is asserted during run-time, the instruction cache treats all misses as if they were from cache-inhibited regions. Misses are loaded only into the burst buffer; hits are loaded from the cache array and the LRU bits are updated. If the debug routine is not in the instruction cache, it is loaded from memory like any other miss and the cache state remains the same as before the freeze signal was asserted.

For performance reasons, it may be preferable to run the debug routine from the cache. To load the debug routine into the instruction cache before entering debug mode, perform the following procedure:

1. Save both ways of the sets that are needed for the debug routine by reading the tag, the LRU, valid, and lock bit states
2. Unlock the locked ways in the selected sets
3. Use the load & lock cache block command to load the debug routine into the instruction cache and lock the cache blocks containing the debug routine.
4. Run the debug routine, all accesses to it will result in hits.

To restore the state of the instruction cache after the debug routine is finished, perform the following procedure:

1. Unlock any ways in any sets that are used by the debug routine
2. Invalidate any ways in any sets that are used by the debug routine
3. Use the load & lock cache block command to restore the old sets in the cache array
4. Unlock any ways of the original sets that were not previously locked
5. To restore the old state of the LRU bits make sure that the last access (load & lock cache block or unlock cache block command) is performed on the most-recently used way (not the LRU way).

When the internal freeze signal is asserted during run-time, the data cache treats all load misses as if they were from cache-inhibited regions. That is, the data is loaded from memory and the cache LRU and state bits are unchanged. Load hits are serviced from the cache array but the cache LRU and state bits are unchanged.

When the internal freeze signal is asserted, store hits and misses are treated as write-through accesses, but the LRU bits in the data cache array are not updated. For the **dcbz** instruction, data is written both into data cache and memory, but the LRU bits in the data cache array are not updated. For the **dcbst**/**dcbf**/**dcbi** instructions, the data cache and memory are updated according to the PowerPC architecture, but the LRU bits in the data cache array are not updated.

Chapter 8
Memory Management Unit

The MPC855T implements a virtual memory management scheme that provides cache control, memory access protections, and effective-to-physical (real) address translation. The MMU largely complies with the PowerPC operating environment architecture (OEA) with respect to architecturally defined memory management features that are appropriate for this implementation. It does not support some PowerPC MMU features more appropriate for a personal computer that is expected to run many applications simultaneously, and in some cases provides greater flexibility than is defined by the PowerPC architecture, especially with respect to page sizes. Available protection granularity is 4-, 16-, 512-Kbyte, or 8-Mbyte pages or 1-Kbyte subpages (for 4-Kbyte pages only). The MPC855T has separate instruction and data MMUs. The prefix M*x*_ indicates a reference to both the instruction and data (MI_ and MD_) versions of the register. The MMU supports two protection modes—default mode with extended encoding and domain manager mode, which provides programmable overrides to page protection settings.

8.1 Features

The following is a list of the MMU's important features:

- Multiple page sizes—4-, 16-, 512-Kbyte, or 8-Mbyte pages (optional 1-Kbyte subpage protection granularity for 4-Kbyte pages) with the following page attributes:
  - Changed bit support through the DTLB error exception on a write attempt to a unmodified page (data MMU only)
  - Write-through attribute for data accesses
  - Cache-inhibit attribute for data and instruction accesses
  - Default write-through and cache-inhibited attributes can be programmed for when translation is disabled
  - Guarded attribute for memory-mapped I/O and other nonspeculative regions
- Instruction and data address translation can be disabled separately
- MPC855T-specific special-purpose registers (SPRs) accessible with the PowerPC **mfspr/mtspr** instructions

- Supports up to 16 virtual address spaces
- Supports 16 access protection groups (group protection overrides page protection)
- Separate -entry, fully-associative data translation lookaside buffer (DTLB) and instruction TLB (ITLB) with the following features:
  - Implementation-specific exceptions—ITLB and DTLB miss exceptions, ITLB and DTLB error exceptions
  - Supports PowerPC **tlbie** and **tlbia** instructions. The **tlbsync** instruction, which is optional to PowerPC architecture implementations, is not supported and is treated as a no-op
  - Software tablewalk updates supported by DTLB and ITLB miss exceptions and SPRs
  - Each entry can be programmed to match user or supervisor accesses or both
  - entries in each TLB can optionally be locked to ensure fast translation for selected regions
- High performance
  - 1 clock (zero wait state) access for a data cache hit and for an instruction cache hit when the access is from the same page as the previous access
  - 1 clock penalty for other TLB hit instruction accesses
- Low power consumption

8.2 PowerPC Architecture Compliance

The MPC855T core complies largely with the MMU as it is defined by the OEA, with the following differences:

- The MPC855T does not implement the following PowerPC features:
  - Block-address translation
  - The optional direct-store functionality
  - The memory coherency attribute
- The MPC855T supports the following additional features not defined by the PowerPC architecture:
  - Variable page sizes. The OEA defines 4-Kbyte pages only
  - Programmable defaults for write-through and cache-inhibited memory attributes when translation is disabled.
  - Additional registers and exceptions for handling table walks in software.

Note that although the MPC855T does not define segment registers as they are defined by the OEA, the concept of segment is retained as the memory space accessible to the level-one table descriptors.

8.3 Address Translation

The core generates 32-bit effective addresses (EA) for memory accesses. Setting MSR[IR] and MSR[DR] enables the effective-to-real translation for instruction fetching and data accesses, respectively. Section 8.3.1, “Translation Disabled,” describes behavior when translation is disabled. Section 8.3.2, “Translation Enabled,” describes behavior when translation is enabled.

8.3.1 Translation Disabled

Because the IMMU and DMMU are separate, translation can be disabled or enabled independently for data and instruction accesses by clearing MSR[DR] and MSR[IR], respectively. When translation is disabled, the effective address is also the physical address.

Because the page translation mechanism is not used, the protection attributes that are part of the page table structure cannot be used, so defaults are used. The default for whether accesses are cache-inhibited are programmed through M*x*_CTR[CIDEF]. Data accesses can be either write-through (memory writes go both to the cache and to external memory) or write back (memory writes directly affect the cache only and memory is updated indirectly, such as when a modified data in the cache is cast-out by newer data at a different address that maps to the same cache block). The default is configured by MD_CTR[WTDEF].

Also, when translation is disabled (real mode), the entire memory space is treated as guarded by default. The implications of this are:

1. Speculative load/store accesses are stalled until they are no longer speculative.
2. Speculative instruction fetches outside of the current real-mode page are stalled until they are no longer speculative. The size of real-mode page is determined by MI_CTR[PPM]. If MI_CTR[PPM] = 0, the real-mode page size is 4 Kbytes; if MI_CTR[PPM] = 1, the real-mode page size is 1 Kbyte.

This behavior can result in significant performance degradation.

8.3.2 Translation Enabled

Translations are generated on a per-page basis and are stored in tables in memory. Along with the translation, each table entry holds attributes for that page, for example, whether a location is cacheable.

Recently used translations are kept in translation lookaside buffers (TLBs) in hardware. In the MPC855T, software handles the table lookup and TLB reload with little hardware assistance. This offers a flexible translation table structure choice, because many systems would not benefit from a full-featured hardware translation mechanism.

A TLB hit in multiple entries is avoided when a TLB is being reloaded. When TLB logic detects that a new effective page number (EPN) overlaps one in the TLB (when taking into account pages sizes, subpage validity flags, user/supervisor state, address pace ID (ASID), and the SH values of the TLB entries), the new EPN is written and the old one is invalidated.

The MMU supports a multiple virtual address space model. Each translation is associated with an ASID, which must equal the address space ID (CASID) for a translation to be valid.

Figure 8-1 shows the flow for a read access or instruction fetch.

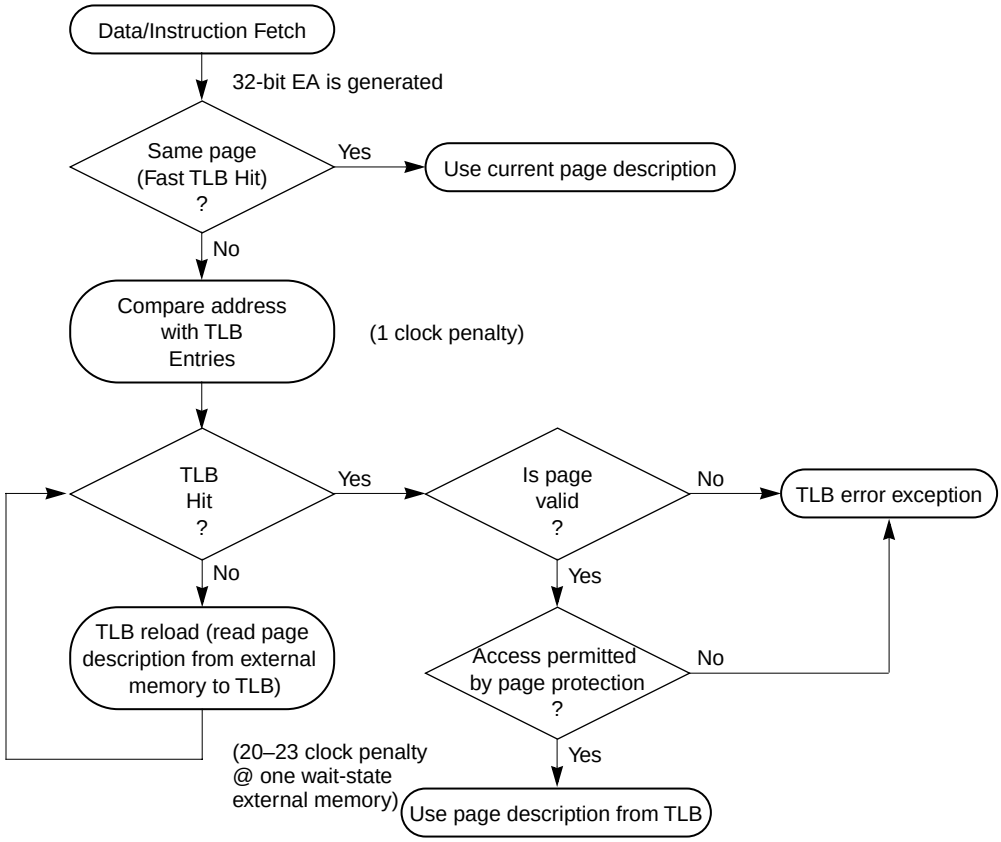


Figure 8-1. Read/Instruction Fetch Flow Diagram

Figure 8-2 shows the flow for a load/store access, assuming translation is enabled. Because data transfers have less locality than instruction fetches, the DMMU does not implement a fast TLB mechanism. The DTLB is accessed for each transfer simultaneously with the data cache tag read, hence there is no time penalty.

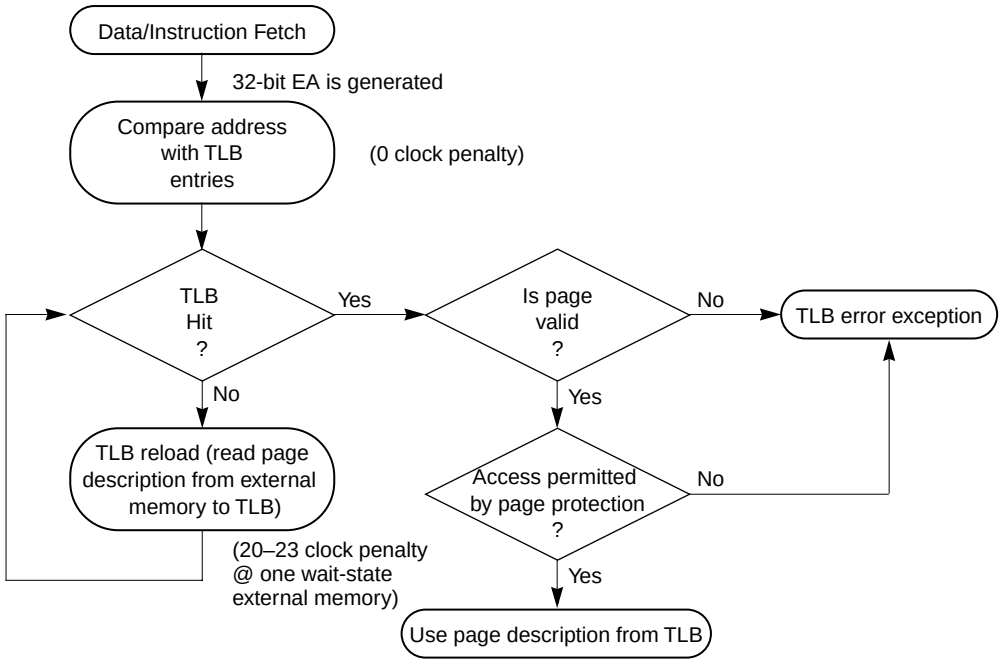


Figure 8-2. Flow of Load/Store Access

8.3.3 TLB Operation

Each TLB contains pointers to pages in physical memory where data is indexed by the EPN. TLBs entries can have different page sizes. The entry page size determines which EA bits are compared and how many of its lsbs pass untranslated as physical address bits.

For a 4-Kbyte page, four subpage validity flags are supported, allowing any combination of 1-Kbyte subpages to be mapped. For any other page size, all of these flags should have the same value. Programming non-4-Kbyte pages with different valid bits is a programming error. Subpage validity flags can be manipulated to implement 1–4 Kbyte pages or any other combination of 1-Kbyte subpages. However, all subpages of an effective page frame must map to the same physical page. During translation, the EA, the privilege level (MSR[PR]), and CASID are provided to the TLB, as shown in Figure 8-3. In the TLB, the EA and CASID are compared with each entry's EPN and ASID. The CASID is compared only when the matching entry is programmed as unshared. See Table 8-12 and Table 8-13.

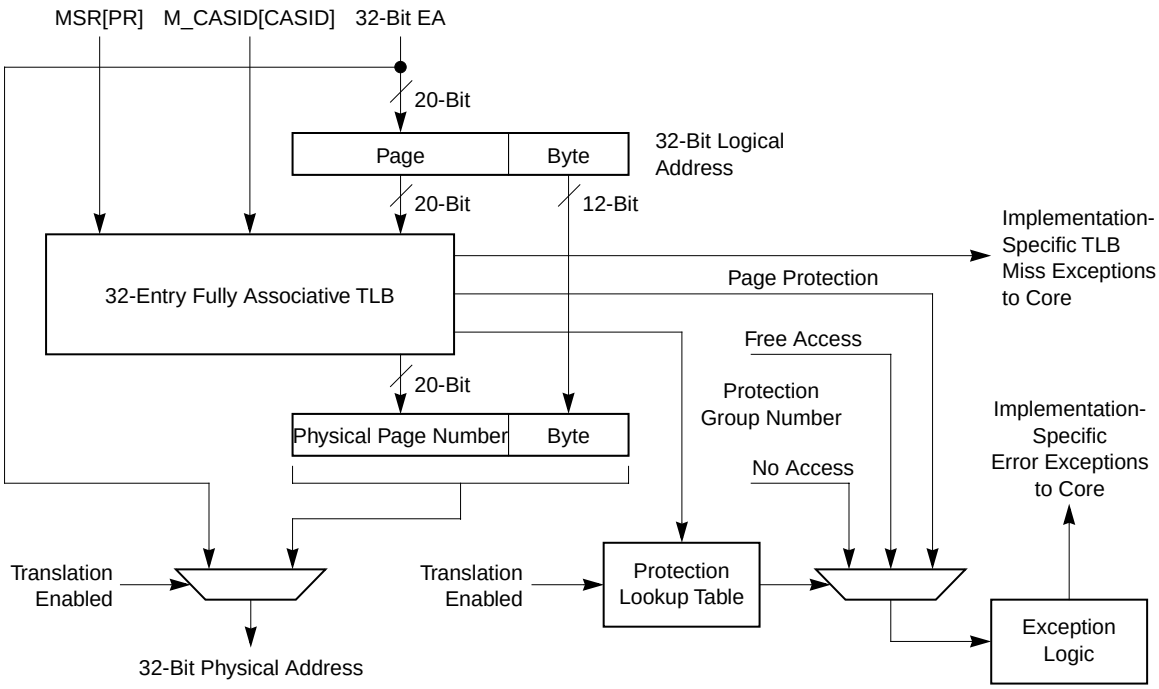


Figure 8-3. Effective-to-Physical Address Translation for 4-Kbyte Pages Block Diagram

A TLB hit occurs if the incoming EA matches the EPN and M_CASID[CASID] matches the ASID field in a valid TLB entry, and if the subpage validity flag is set for the subpage that the incoming EA points to. If a hit is detected, the contents of the physical page number are concatenated with the appropriate number of lsbs from the EA to form the physical address sent to the cache and memory system.

8.4 Using Access Protection Groups

Access control is assigned on a page-by-page basis; additional control is provided on a protection group basis. 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 (MI_AP or MD_AP) that defines access control for the translation. Each M*x*_AP contains 16 fields. The field content is used according to the group protection mode.

To be consistent with the PowerPC OEA, the APG value should match the four msbs of the effective page number. To override protection using APG, use it on blocks of addresses which are defined by the 4 msbs of the effective page number. If APG is not to be used for a particular block, set the GP for that block to 'client' in the M*x*_AP register. To ignore it globally, set all of the M*x*_AP fields to 01. In default mode, each field holds the Kp and Ks bits for the corresponding segment defined by the level-one table descriptor. In domain

manager mode, each field holds override information over the page protection setting—no override, no access override, and free access override.

8.5 Protection Resolution Modes

The MMUs can be programmed in three different modes that have different methods of defining the protection resolution of the address space. These are as follows:

- Mode 1—Protection resolution to 4-Kbyte minimum page size. This is the simplest mode with the most efficient memory size (that is, MMU tables are smaller). Use this mode if 1-Kbyte protection resolution is not required.

  In this mode, program the following:

  — MD_CTR[TWAM] = 1

  — Mx_CTR[PPM] = 0

  — Bits 20–27 of the level-two descriptor take on the meaning described in the right side of Table 8-4.

- Mode 2—Protection resolution to 1-Kbyte minimum subpage size, where all 4-Kbyte logical address pages map to the same 4-Kbyte physical page, but the four 1-Kbyte subpages may have different protection attributes.

  In this mode, program the following:

  — MD_CTR[TWAM]=1

  — Mx_CTR[PPM]=1

  For 4-Kbyte pages, program the four PP pairs (bits 20–27) to the subpage protection attributes for the 1-Kbyte subpages.

  For pages larger than 4 Kbytes, the four PP pairs (bits 20–27) must all be programmed to the same protection attributes, which are applied to the full page.

  This mode is just as efficient in memory size as mode 1, but has the memory protection resolution of mode 3.

- Mode 3—Protection resolution to 1-Kbyte minimum subpage size, with no restriction on subpage mapping. In this mode, program:

  — MD_CTR[TWAM] = 0

  — Mx_CTR[PPM] = 0

  — Mx_CTR[PPCS] = 0

  For pages larger than 4-Kbyte, program subpage validity flags (bits 24-27) of the level-two descriptor (and thus M*x*_RPN) to 0b1111.

For 4-Kbyte pages, there are four separate entries with different encodings of subpage validity flags (bits 24–27) of the level-two descriptor (and thus Mx_RPN) allowable for each entry.

For 4-Kbyte pages, the subpage validity flags (bits 24–27) of the level-two descriptor (and thus Mx_RPN) can be different for each of the four separate entries.

In this mode, the MMU page tables defined for the software tablewalk resolve to a single level-two descriptor entry for a 1-Kbyte page. This is done by allowing manipulation of the subpage validity flags of a 4-Kbyte page. For example:

— To define a 4-Kbyte page with uniform protection, create four level-two descriptors for the 4-Kbyte page, each with subpage validity flags set to 0b1111. All other fields of the level-two descriptors must also be the same for each of these entries.

— To define four different 1-Kbyte pages, create four level-two descriptors, but set the subpage validity flags such that: entry one = 0b1000, entry two = 0b0100, entry three = 0b0010, entry four = 0b0001. All other fields of the level-two descriptor can be set differently for each of these entries.

— To define two different 2-Kbyte pages, create four level-two descriptors, but set the subpage validity flags in pairs such that: entry one = 0b1100, entry two = 0b1100, entry three = 0b0011, entry four = 0b0011. The other fields of the ‘paired’ level-two descriptors must be the same for each of the pairs.

  Other combinations are also possible.

This mode is the most complex and the most inefficient in memory size (that is, MMU tables are approximately four times larger). However, it allows the most detailed resolution of protection with full functionality.

IMMUs and DMMUs can use different modes; the IMMU could use mode 1 and the DMMU could use mode 2, or vice versa. However, if mode 3 is desired, both MMUs must be in mode 3.

8.6 Memory Attributes

Memory attributes defined by the PowerPC architecture are implemented as follows:

- Reference and change bit updates—The MPC855T does not generate an exception for an R (reference) bit update. In fact, there is no entry for an R bit in the TLB.

  The change bit (C) is bit 23 in the level-two descriptor, described in Table 8-4. Software updates C (changed) bits, but hardware treats the C bit (negated) as a write-protect attribute. Therefore, attempting to write to a page marked unmodified invalidates that entry and causes an implementation-specific DTLB error exception. If change bits are not needed, set the C bit to one by default in the PTEs.

- Memory control attributes—The MPC855T supports cache inhibit (CI), writethrough (WT), and guarded (G) attributes, defined in the PowerPC Virtual Environment Architecture (VEA). The memory coherence (M) attribute is not supported; to ensure memory coherency, configure the page as cache-inhibited. Chapter 7, “Instruction and Data Caches,” describes the effects of CI and WT attributes in the MPC855T.

  The G attribute is used to map I/O devices that are sensitive to speculative (out-of-order) accesses. An attempted speculative access to a page marked guarded (G = 1) stalls until either the access is nonspeculative or is canceled by the core. Attempting to fetch from guarded memory causes an implementation-specific instruction TLB error interrupt.

8.7 Translation Table Structure

The MMU hardware supports a two-level software tablewalk. Other table structures are not precluded. Figure 8-4 shows the two-level translation table when MD_CTR[TWAM] = 1 (4-Kbyte resolution of protection).

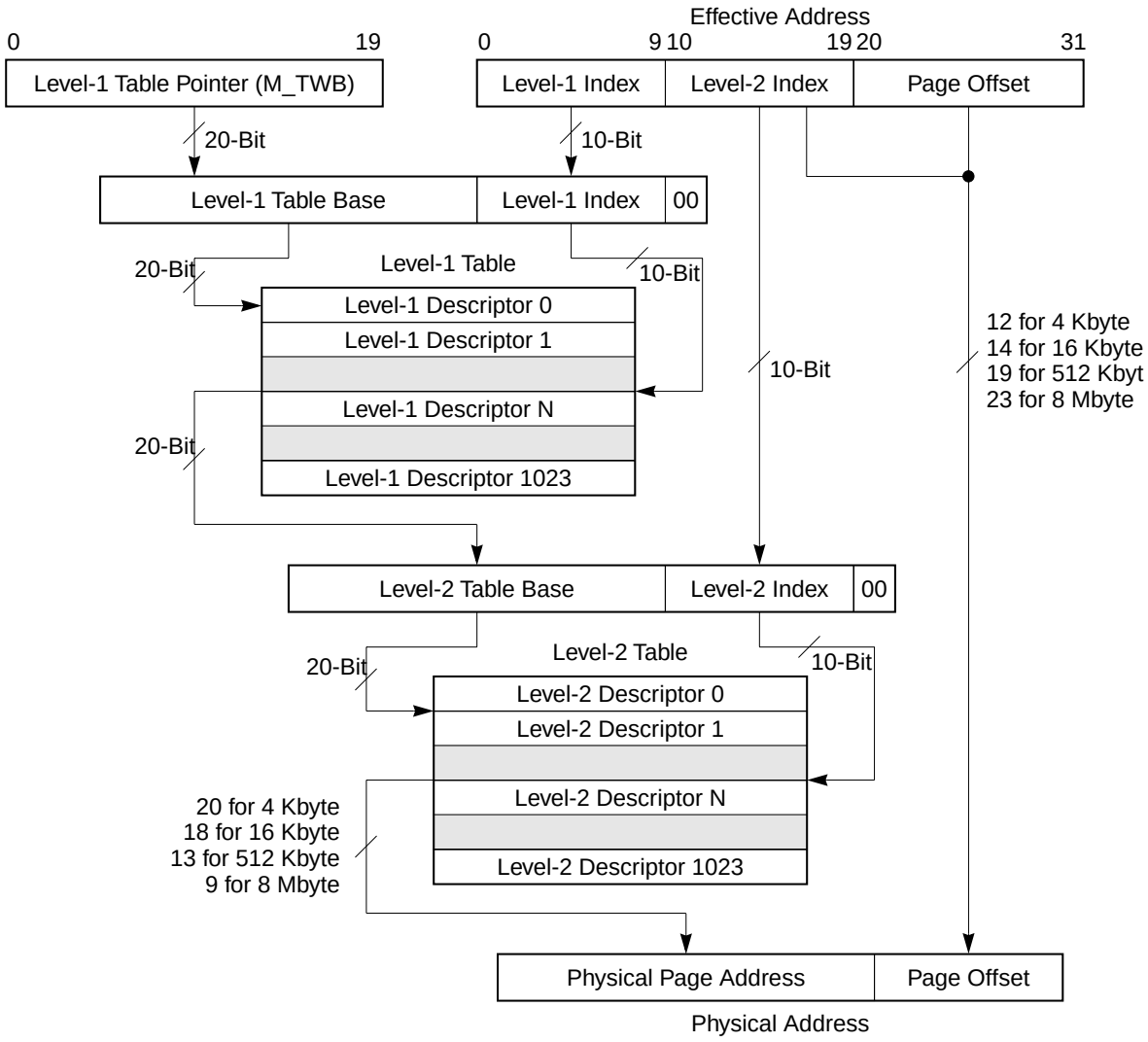


Figure 8-4. Two-Level Translation Table (MD_CTR[TWAM] = 1)

When MD_CTR[TWAM] = 1, the tablewalk begins at the level-one base address in M_TWB. EA[0–9] indicates the level-one page descriptor. As shown in Table 8-1, an 8-Mbyte page requires two identical entries in the level-one table, one for bit 9 = 0 and one for bit 9 = 1.

Table 8-1. Identical Entries Required in Level-One/Level-Two Tables

Page Size	Identical Entries Required in Level-One Table		Identical Entries Required in Level-Two Table	
	MD_CTR[TWAM] = 0	MD_CTR[TWAM] = 1	MD_CTR[TWAM] = 0	MD_CTR[TWAM] = 1
1 Kbyte	1	—	1	—
4 Kbyte	1	1	4	1
16 Kbyte	1	1	16	4
512 Kbyte	1	1	1	1
8 Mbyte	8	2	1	1

The page size and the level-two base address are read from the level-one descriptor. If the page size is 512 Kbytes or 8Mbytes, the level-two base address is used directly as the address of the level-two descriptor. If the page size is less than 512 Kbytes, the address of the level-two descriptor is determined by indexing the level-two base address by EA[10–19]. For 16 Kbyte pages, this requires that multiple identical level-two descriptors be provided. This is summarized in Table 8-1.

Figure 8-5 shows the two-level translation table when MD_CTR[TWAM] = 0 (1 Kbyte resolution of protection).

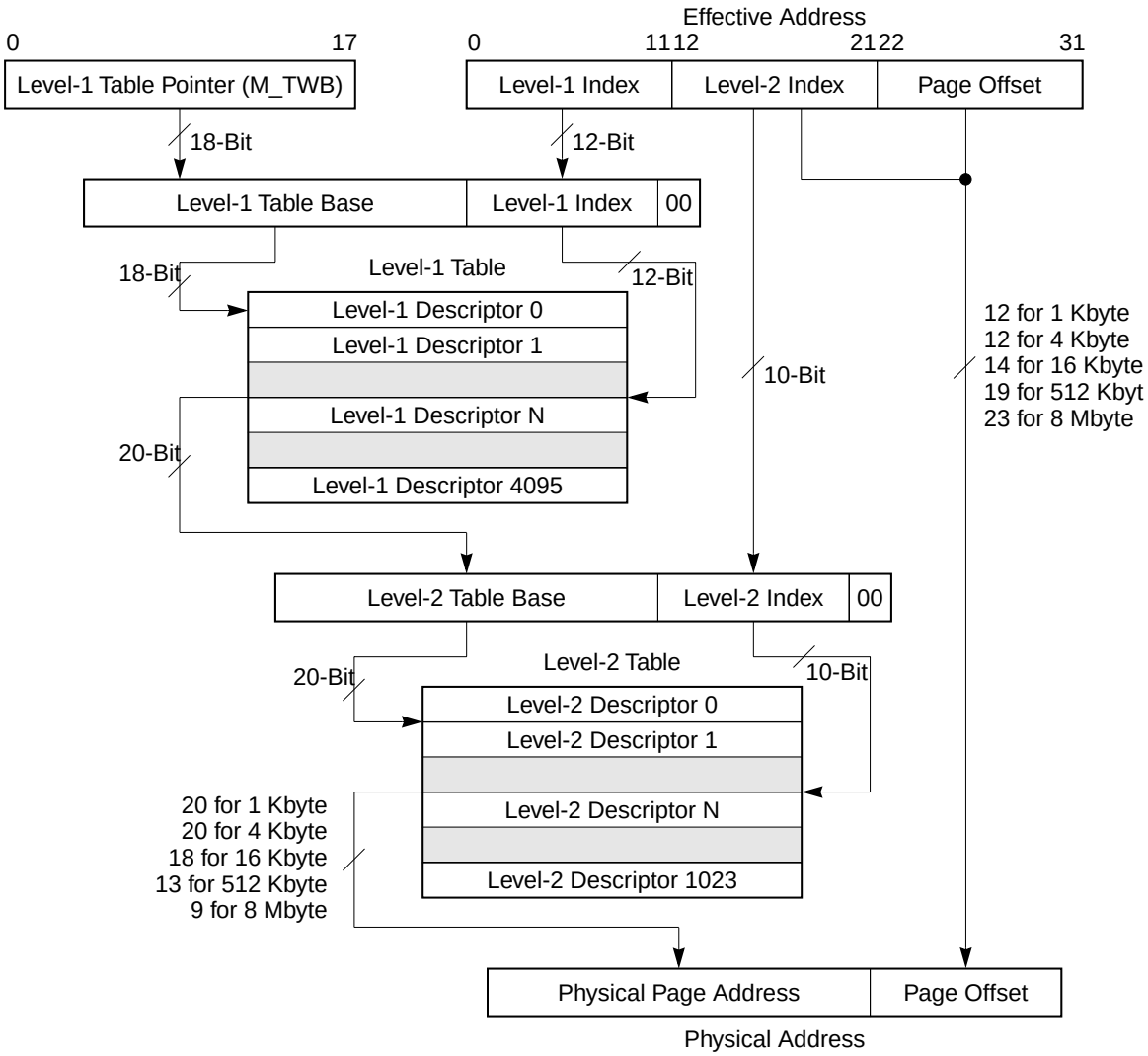


Figure 8-5. Two-Level Translation Table (MD_CTR[TWAM] = 0)

During address translation, the msbs of the missed effective address are replaced by the physical page address bits from the level-two page descriptor; the page size determines the number of replaced bits as shown in Table 8-2. The remaining physical address bits come directly from the effective address. When MD_CTR[TWAM] = 0, the tablewalk begins at the level-one base address placed in M_TWB. The level-one table is indexed by EA[0–11] to get the level-one page descriptor. As shown in Table 8-1, 8-Mbyte pages must have eight identical entries in the level-one table for EA[9–11].

Table 8-2. Number of Replaced EA Bits per Page Size

Page Size	Number of Replaced EA Bits
1 Kbyte	20
4 Kbyte	20
16 Kbyte	18
512 Kbyte	13
8 Mbyte	9

The page size and the level-two base address are read from the level-one descriptor. If the page size is 512 Kbytes or 8Mbytes, the level-two base address is used directly as the address of the level-two descriptor. If the page size is less than 512 Kbytes, the address of the level-two descriptor is determined by indexing the level-two base address by EA[12–21]. For 4Kbyte or 16 Kbyte pages, this requires that multiple identical level-two descriptors be provided. This is summarized in Table 8-1.

The number of replaced bits depends on the page size, as shown in Table 8-2. The remaining physical address bits are taken directly from the effective address.

8.7.1 Level-One Descriptor

Table 8-3 describes the level-one descriptor format supported by the hardware to minimize the software tablewalk routine.

Table 8-3. Level-One Segment Descriptor Format

Bits	Name	Description
0–19	L2BA	Level-2 table base address. Bits 18–19 should be 0b00 unless MD_CTR[TWAM] = 1.
20–22	—	Reserved
23–26	APG	Access protection group
27	G	Guarded memory attribute for entry 0 Nonguarded memory 1 Guarded memory
28–29	PS	Page size level one. Used with the level-two (L2) descriptor's small-page-size (SPS) field; see Section 8.7.3, "Page Size." 00 Small (4 Kbyte or 16 Kbyte) 01 512 Kbyte 10 Reserved 11 8 Mbyte
30	WT	Writethrough attribute for entry 0 Copyback cache policy region (default) 1 Writethrough cache policy region
31	V	Level-one segment valid bit 0 Segment is not valid 1 Segment is valid

8.7.2 Level-Two Descriptor

Table 8-4 describes the level-two descriptor format supported by hardware. (Section 8.5, “Protection Resolution Modes,” describes the protection modes.)

Table 8-4. Level-Two (Page) Descriptor Format

<table>
<tr><th>Bits</th><th>Name</th><th colspan="2">Mode 2</th><th colspan="2">Mode 1 or Mode 3</th></tr>
<tr><td>0–19</td><td>RPN</td><td colspan="4">Physical (real) page number</td></tr>
<tr><td>20–21</td><td>PP</td><td>Protection for 1st subpage</td><td rowspan="5">For Instruction Pages
Supervisor User
00 No access No access
01 Executable No access
1x Executable Executable

For Data Pages
Supervisor User
00 No access No access
01 R/W No access
10 R/W R/O
11 R/W R/W</td><td>For Instruction Pages
Supervisor User
Extended encoding:
00 No access No access
01 Executable No access
1x Reserved
Basic encoding:
00 Executable No access
01 Executable Executable
1x Executable Executable</td><td>For Data Pages
Supervisor User
Extended encoding:
00 No access No access
01 R/O No access
1x Reserved
Basic encoding:
00 R/W No access
01 R/W R/O
10 R/W R/W
11 R/O R/O</td></tr>
<tr><td>22</td><td>PP [1]</td><td>2nd subpage</td><td colspan="2">0 Bits 20–21 contain Basic encoding
1 Bits 20–21 contain extended encoding</td></tr>
<tr><td>23</td><td></td><td></td><td colspan="2">C—Change bit for entry. Set to 1 by default if change tracking functionality is not desired.
0 Unchanged region (write-protected)
1 Changed region, write allowed</td></tr>
<tr><td>24–25</td><td rowspan="2"></td><td>3rd subpage</td><td rowspan="2">MD_CTR[PPCS] = 0.
For 1-Kbyte pages in mode 3, program to the appropriate subpage validity. For mode 1, program to 0b1111.</td><td rowspan="2">MD_CTR[PPCS] = 1 (mode 1 only)
1000 Hit (supervisor accesses only)
0100 Hit (user accesses only)
1100 Hit for both</td></tr>
<tr><td>26–27</td><td>4th subpage</td></tr>
<tr><td>28</td><td>SPS</td><td colspan="4">Small page size. Used with the level-one (L1) descriptor’s page-size (PS) field; see Section 8.7.3, “Page Size.”
0 4 Kbyte
1 16 Kbyte or larger (512 Kbyte or 8 Mbyte)</td></tr>
<tr><td>29</td><td>SH</td><td colspan="4">Shared page
0 Entry matches only if a TLB entry’s ASID field matches the value in M_CASID.
1 ASID comparison is disabled for the entry.</td></tr>
<tr><td>30</td><td>CI</td><td colspan="4">Cache-inhibit attribute for the entry.
0 Caching is allowed.
1 Caching is inhibited.</td></tr>
<tr><td>31</td><td>V</td><td colspan="4">Page valid bit</td></tr>
</table>

[1] For pages larger than 4 Kbytes in mode 2, PP in bits [22–23,24–25,26–27] must equal the PP in bits [20–21].

8.7.3 Page Size

The page size is determined by a combination of two fields: the page-size (PS) field in the level-one descriptor and the small-page-size (SPS) field in the level-two descriptor. Table 8-5 shows how the two fields select the page size.

Table 8-5. Page Size Selection

Level 1 [PS]	Level 2 [SPS]	Page Size
00	0	4 Kbyte
00	1	16 Kbyte
01	0	reserved
01	1	512 Kbyte
10	x	reserved
11	0	
11	1	8 Mbyte

8.8 Programming Model

All MMU programming model registers are supervisor-level SPRs that are accessed by using **mtspr** and **mfspr**. Attempting to access these SPRs in user mode causes a program exception. The **tlbie** and **tlbia** instructions can be used to invalidate TLBs. MMU registers should be accessed when both MSR[IR] = 0 and MSR[DR] = 0. No similar restriction exists for **tlbie** and **tlbia**.

Table 8-6 lists the MPC855T-specific MMU registers and indicates the sections that describe them. These SPRs should be accessed when both instruction and data address translation is disabled.

Table 8-6. MPC855T-Specific MMU SPRs

Register	Name	SPR	Section
Control Registers			
MI_CTR	IMMU control register	784	8.8.1
MD_CTR	DMMU control register	792	8.8.2
TLB Source Registers			
MI_EPN	IMMU effective number register	787	8.8.3
MD_EPN	DMMU effective number register	795	
MI_TWC	IMMU tablewalk control register	789	8.8.4
MD_TWC	DMMU tablewalk control register	797	8.8.5
MI_RPN	IMMU real (physical) page number port	790	8.8.6
MD_RPN	DMMU real (physical) page number register	798	8.8.7

Table 8-6. MPC855T-Specific MMU SPRs (continued)

Register	Name	SPR	Section
Tablewalk Assist Registers			
M_TWB	MMU tablewalk base register	796	8.8.8
Protection Registers			
M_CASID	CASID register	793	8.8.9
MI_AP	IMMU access protection register	786	8.8.10
MD_AP	DMMU access protection register	794	
Scratch Register			
M_TW	MMU tablewalk special register	799	8.8.11
Debug Registers			
MI_CAM	IMMU CAM entry read register	816	8.8.12.1
MI_RAM0	IMMU RAM entry read register 0	817	8.8.12.2
MI_RAM1	IMMU RAM entry read register 1	818	8.8.12.3
MD_CAM	DMMU CAM entry read register	824	8.8.12.4
MD_RAM0	DMMU RAM entry read register 0	825	8.8.12.5
MD_RAM1	DMMU RAM entry read register 1	826	8.8.13

8.8.1 IMMU Control Register (MI_CTR)

The IMMU control register (MI_CTR), shown in Figure 8-6, controls IMMU operation.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	GPM	PPM	CIDEF	—	RSV4I	—	PPCS	—								
Reset	0															
R/W	R/W															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—			ITLB_INDX					—							
Reset	0															
R/W	R/W															
SPR	784															

Figure 8-6. IMMU Control Register (MI_CTR)

Table 8-7 describes MI_CTR fields.

Table 8-7. MI_CTR Field Descriptions

Bits	Name	Description
0	GPM	Group protection mode 0 Default mode 1 Domain manager mode
1	PPM	Page protection mode. Valid regardless of whether translation is enabled. If translation is enabled, PPM determines how M*x*_RPN is interpreted. See Table 8-12 and Table 8-13. 0 Page resolution of protection 1 1-Kbyte resolution of protection for 4-Kbyte pages
2	CIDEF	Default value for instruction cache-inhibit attribute when the IMMU is disabled (MSR[IR] = 0) 0 Caching is allowed. 1 Caching is inhibited.
3	—	Reserved. Ignored on write. Returns 0 on read.
4	RSV4I	Reserve four ITLB entries. See Section 8.10.2, “Locking TLB Entries.” 0 ITLB_INDX decremented modulo 32 1 ITLB_INDX decremented modulo 28
5	—	Reserved. Ignored on write. Returns 0 on read.
6	PPCS	Privilege/user state compare mode 0 Ignore user/supervisor state during address compare 1 Account for user/supervisor state according to MI_RPN[24–27]
7–18	—	Reserved. Ignored on write. Returns 0 on read.
19–23	ITLB_INDX	ITLB index. Points to the ITLB entry to be loaded. Decremented every ITLB update
24–31	—	Reserved. Ignored on write. Returns 0 on read.

8.8.2 DMMU Control Register (MD_CTR)

The DMMU control register (MD_CTR), shown in Figure 8-7, controls DMMU operation.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	GPM	PPM	CIDEF	WTDEF	RSVD	TWAM	PPCS	—								
Reset	0000_0					1	0	0_0000_0000								
R/W	R/W															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—			DTLB_INDX					—							
Reset	0x0000															
R/W	R/W															
SPR	792															

Figure 8-7. DMMU Control Register (MD_CTR)

Table 8-8 describes MD_CTR fields.

Table 8-8. MD_CTR Field Descriptions

Bits	Name	Description
0	GPM	Group protection mode 0 Default mode 1 Domain manager mode
1	PPM	Page protection mode 0 Page resolution of protection 1 1-Kbyte resolution of protection for 4-Kbyte pages
2	CIDEF	CI default when the DMMU is disabled (MSR[DR] = 0) 0 Caching is allowed. 1 Caching is inhibited.
3	WTDEF	WT default when the DMMU is disabled (MSR[DR] = 0)
4	RSV4D	four Reserve two DTLB entries. See Section 8.10.2, “Locking TLB Entries.” 0 DTLB_INDX decremented modulo 32 1 DTLB_INDX decremented modulo 28
5	TWAM	Tablewalk assist mode 0 1-Kbyte subpage hardware assist 1 4-Kbyte page hardware assist (default)
6	PPCS	Privilege/user state compare mode 0 Ignore user/supervisor state during address compare 1 Account for user/supervisor state according to MD_RPN[24–27]
7–18	—	Reserved. Ignored on write. Returns 0 on read
19–23	DTLB_INDX	DTLB index. Points to DTLB entry to be loaded. Decremented every DTLB update.
24–31	—	Reserved. Ignored on write. Returns 0 on read

8.8.3 IMMU/DMMU Effective Page Number Register (M*x*_EPN)

The effective page number registers (MI_EPN and MD_EPN), shown in Figure 8-8, contain the EA to be loaded into a TLB entry.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	EPN															
Reset	0000_0000_0000_0000															
R/W	R/W															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	EPN				—		EV	—					ASID			
Reset	—				0		0	0					0			
R/W	R/W				R		R/W	R					R/W			
SPR	787 (MI_EPN); 795 (MD_EPN)															

Figure 8-8. IMMU/DMMU Effective Page Number Register (M*x*_EPN)

Table 8-9 describes Mx_EPN fields.

Table 8-9. M*x*_EPN Field Descriptions

Bits	Name	Description
0–19	EPN	Effective page number for TLB entry. Default value is the EA of the last ITLB/DTLB miss
20–21	—	Reserved. Ignored on write. Undefined on read
22	EV	TLB entry valid bit. 0 TLB entry is invalid 1 TLB entry is valid. EV is set to 1 on each ITLB/DTLB miss.
23–27	—	Reserved. Ignored on write. Returns 0 on read
28–31	ASID	Address space ID of the ITLB/DTLB entry to be compared with M_CASID[CASID]. Loaded with M_CASID on a TLB miss.

8.8.4 IMMU Tablewalk Control Register (MI_TWC)

The IMMU tablewalk control register (MI_TWC), shown in Figure 8-9, contains the access protection group and page size of the entry to be loaded into the TLB.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Reset	0000_0000_0000_0000															
R/W	R/W															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—							APG				G	PS		—	V
Reset	0							—				—	—		0	—
R/W	R/W							R/W				R/W	R/W		R/W	R/W
SPR	789															

Figure 8-9. IMMU Tablewalk Control Register (MI_TWC)

Table 8-10 describes MI_TWC fields.

Table 8-10. MI_TWC Field Descriptions

Bits	Name	Description
0–22	—	Reserved. Ignored on write. Returns 0 on read.
23–26	APG	Access protection group. Up to 16 protection groups supported. Default for ITLB miss is 0
27	G	Guarded memory attribute for entry 0 Nonguarded memory (default for ITLB miss) 1 Guarded memory

Table 8-10. MI_TWC Field Descriptions (continued)

Bits	Name	Description
28–29	PS	Page size level-one 00 Small (4 or 16 Kbyte. See MI_RPN[SPS]) Default for ITLB miss 01 512 Kbyte 10 Reserved 11 8 Mbyte
30	—	Reserved. Ignored on write. Returns 0 on read.
31	V	Entry valid bit 0 Entry is not valid 1 Entry is valid. Default value on ITLB miss.

8.8.5 DMMU Tablewalk Control Register (MD_TWC)

The DMMU tablewalk control register (MD_TWC), shown in Figure 8-10, contains the level-two pointer and access protection group of an entry to be loaded into the TLB.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	L2TB															
Reset	—															
R/W	R/W															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	L2TB				—			APG				G	PS		WT	V
Reset	—															
R/W	R/W															
SPR	797															

Figure 8-10. DMMU Tablewalk Control Register (MD_TWC)

Table 8-11 describes MD_TWC fields.

Table 8-11. MD_TWC Field Descriptions

Bits	Name		Description	
	Write	Read	Write	Read
0–19	L2TB	L2TB	Tablewalk level-two table base value	

Table 8-11. MD_TWC Field Descriptions (continued) (continued)

Bits	Name		Description	
	Write	Read	Write	Read
20–22	—	L2INDX	Ignore	Level-two table index. Returns MD_EPN[10–19] when MD_CTR[TWAM] = 1 Returns MD_EPN[12–21] when MD_CTR[TWAM] = 0
23–26	APG		Access protection group. Up to 16 protection groups are supported. Set to 0000 on a DTLB miss.	
27	G		Guarded memory attribute of the entry: 0 Nonguarded memory. Cleared on DTLB miss. 1 Guarded memory	
28–29	PS		Level-one page size. (Cleared on a DTLB miss.) 00 Small (4 Kbyte or 16 Kbyte. See MD_RPN) 01 512 Kbyte 10 Reserved 11 8 Mbyte	
30	WT	—	Writethrough attribute for page entry: 0 Copyback data cache policy. Cleared on DTLB miss. 1 Writethrough data cache policy	Returns 0 on read.
31	V	—	0 Entry is not valid 1 Entry is valid. (set on a DTLB miss)	Returns 0 on read.

8.8.6 IMMU Real Page Number Register (MI_RPN)

The IMMU real page number register (MI_RPN), shown in Figure 8-11, contains the physical address and the memory attributes of an entry to be loaded into a TLB. MI_RPN should be written after MI_EPN and MI_TWC are written.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	RPN															
Reset	—															
R/W	R/W															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	RPN				PP								SPS	SH	CI	V
Reset	—															
R/W	R/W															
SPR	790															

Figure 8-11. IMMU Real Page Number Register (MI_RPN)

Table 8-12 describes MI_RPN fields. (Section 8.5, “Protection Resolution Modes,” describes the protection modes.)

Table 8-12. MI_RPN Field Descriptions

<table>
<tr><th>Bits</th><th>Name</th><th>Mode 2</th><th colspan="2">Mode 1 or Mode 3</th></tr>
<tr><td>0–19</td><td>RPN</td><td colspan="3">Real (physical) page number</td></tr>
<tr><td>20–21</td><td>PP</td><td rowspan="6">Protection attributes for subpages 1–4.
Supervisor User
00 No access No access
01 Executable No access
1x Executable Executable</td><td colspan="2">Extended Encoding:
Supervisor User
00 No access No access
01 Executable No access
1x Reserved Reserved
Basic Encoding:
Supervisor User
00 Executable No access
01 Executable Executable
1x Executable Executable</td></tr>
<tr><td>22</td><td rowspan="4">PP [1]</td><td colspan="2">0 Bits 20–21 contain Basic encoding
1 Bits 20–21 contain extended encoding</td></tr>
<tr><td>23</td><td colspan="2">Reserved</td></tr>
<tr><td>24–25</td><td rowspan="2">MD_CTR[PPCS] = 0
For 1 Kbyte pages in mode 3, set to the appropriate subpage validity. Otherwise, set to 0b1111.</td><td rowspan="2">MD_CTR[PPCS] = 1
1000 Hit only for supervisor accesses
0100 Hit only for user accesses
1100 Hit for both</td></tr>
<tr><td>26–27</td></tr>
<tr><td>28</td><td>SPS</td><td colspan="3">Small page size. Used with the level-one (L1) descriptor’s page-size (PS) field; see Section 8.7.3, “Page Size.”
0 4 Kbyte
1 16 Kbyte or larger (512 Kbyte or 8 Mbyte)</td></tr>
<tr><td>29</td><td>SH</td><td colspan="3">Shared page:
0 This entry matches only if ASID field in the TLB entry matches the value M_CASID.
1 ASID comparison is disabled for the entry.</td></tr>
<tr><td>30</td><td>CI</td><td colspan="3">Cache-inhibit attribute for the entry.
0 Caching is allowed.
1 Caching is inhibited.</td></tr>
<tr><td>31</td><td>V</td><td colspan="3">Entry valid indication.</td></tr>
</table>

[1] For pages larger than 4 Kbytes in mode 2, PP in bits [22–23,24–25,26–27] must equal the PP in bits [20–21].

8.8.7 DMMU Real Page Number Register (MD_RPN)

The DMMU real page number register (MD_RPN), shown in Figure 8-12, contains the physical address and the memory attributes of an entry to be loaded into a TLB. This register should be written after the MD_EPN and MD_TWC registers.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	RPN															
Reset	—															
R/W	R/W															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	RPN				PP								SPS	SH	CI	V
Reset	—															
R/W	R/W															
SPR	798															

Figure 8-12. DMMU Real Page Number Register (MD_RPN)

Table 8-13 describes MD_RPN fields. (Section 8.5, “Protection Resolution Modes,” describes the protection modes.)

Table 8-13. MD_RPN Field Descriptions

Bits	Name	Mode 2	Mode 1 or Mode 3	
0–19	RPN	Real (physical) page number		
20–21	PP	Protection attributes for subpages 1–4. Supervisor User 00 No access No access 01 R/W No access 10 R/W R/O 11 R/W R/W	Extended Encoding: Supervisor User 00 No access No access 01 R/O No access 1x Reserved	Basic Encoding: Supervisor User 00 R/W No access 01 R/W R/O 10 R/W R/W 11 R/O R/O
22	PP [1]		0 Bits 20–21 contain Basic encoding 1 Bits 20–21 contain extended encoding	
23			Change bit for DTLB entry. Set to 1 by default if change tracking functionality is not desired. 0 Unchanged region. Write access causes an IMMU exception. Software should take an appropriate action before setting this bit. 1 Changed region. Write access is allowed to this page.	
24–25 26–27			MD_CTR[PPCS] = 0 For 1 Kbyte pages in mode 3, set to the appropriate subpage validity. Otherwise, set to 0b1111.	MD_CTR[PPCS] = 1 1000 Hit only for supervisor accesses 0100 Hit only for user accesses 1100 Hit for both
28	SPS	Small page size. Used with the level-one (L1) descriptor’s page-size (PS) field; see Section 8.7.3, “Page Size.” 0 4 Kbyte 1 16 Kbyte or larger (512 Kbyte or 8 Mbyte)		
29	SH	Shared page 0 This entry matches only if the ASID field in the DTLB entry matches the M_CASID value. 1 ASID comparison is disabled for the entry.		

Table 8-13. MD_RPN Field Descriptions (continued)

Bits	Name	Mode 2	Mode 1 or Mode 3
30	CI	Cache-inhibit attribute for the entry. 0 Caching is allowed. 1 Caching is inhibited.	
31	V	Entry valid indication.	

1 For pages larger than 4 Kbytes in mode 2, PP in bits [22–23,24–25,26–27] must equal the PP in bits [20–21].

8.8.8 MMU Tablewalk Base Register (M_TWB)

The MMU tablewalk base register (M_TWB), shown in Figure 8-13, contains a pointer to the level-one table to be used in hardware-assisted tablewalk mode.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	L1TB																L1TB				L1INDX										—	
Reset	—																														00	
R/W	R/W																															
SPR	796																															

Figure 8-13. MMU Tablewalk Base Register (M_TWB)

Table 8-14 describes M_TWB fields.

Table 8-14. M_TWB Field Descriptions

Bits	Name	Description
0–19	L1TB	Tablewalk level-one base value
20–29	L1INDX	Level-one table index. Ignored on write. Returns MD_EPN[0–9] on read when MD_CTR[TWAM] = 1. Returns MD_EPN[2–11] on read when MD_CTR[TWAM] = 0
30–31	—	Reserved. Ignored on write. Returns 0 on read.

8.8.9 MMU Current Address Space ID Register (M_CASID)

The MMU current address space ID register (M_CASID), shown in Figure 8-14, is used to compare the current EA with the ASID field in the TLB entry when searching for a match.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—																												CASID			
Reset	—																															
R/W	R/W																															
SPR	793																															

Figure 8-14. MMU Current Address Space ID Register (M_CASID)

Table 8-15 describes M_CASID fields.

Table 8-15. M_CASID Field Descriptions

Bits	Name	Description
0–27	—	Reserved. Ignored on write. Returns 0 on read
28–31	CASID	Current address space ID. Compared with ASID field of a TLB entry to qualify a match

8.8.10 MMU Access Protection Registers (MI_AP/MD_AP)

The IMMU access protection register (MI_AP, SPR 786) contains the settings for the access protection groups for the IMMU. The DMMU access protection register (MD_AP, SPR 794) is identical. Both registers are shown in Figure 8-15.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	GP0		GP1		GP2		GP3		GP4		GP5		GP6		GP7		GP8		GP9		GP10		GP11		GP12		GP13		GP14		GP15	
Reset	—																															
R/W	R/W																															
SPR	786 (MI_AP); 794 (MD_AP)																															

Figure 8-15. MMU Access Protection Registers (MI_AP/MD_AP)

MI_AP/MD_AP fields are described in Table 8-16.

Table 8-16. MI_AP/MD_AP Field Descriptions

Bits	Name	Domain Manager Mode (M*x*_CTR[GPM] = 1)	Default Mode (M*x*_CTR[GPM] = 0)
0–1	GP*x*	GP 00 No access 01 Client–access permission defined by page protection bits 10 Reserved 11 Manager–free access	GP = Ks/Kp as defined by PowerPC architecture 00 All accesses are treated as supervisor 01 Access permission defined by page protection bits 10 User and supervisor interpretation is swapped 11 All accesses are treated as user
2–3			
…			
30–31			

8.8.11 MMU Tablewalk Special Register (M_TW)

The MMU tablewalk special register (M_TW), shown in Figure 8-16, is a scratch register used by tablewalk exception handlers.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Reset	—																															
R/W	R/W																															
SPR	799																															

Figure 8-16. MMU Tablewalk Special Register (M_TW)

8.8.12 MMU Debug Registers

The MMU CAM and RAM entries can be read through MX_CAM, MX_RAM0, and MX_RAM1. Attempting to write to MX_CAM using an **mtspr** instruction loads the CAM and RAM values of the entry indexed by DTLB_INDX to MX_CAM, MX_RAM0, and MX_RAM1. Any register can be the source for **mtspr** since its value is not used. The values of MX_CAM, MX_RAM0, and MX_RAM1 can be read using **mfspr**; **mtspr**[MX_RAM0] and **mtspr**[MX_RAM1] are considered no-ops.

8.8.12.1 IMMU CAM Entry Read Register (MI_CAM)

Figure 8-17 shows the MMU instruction CAM entry read register (MI_CAM). When the content-addressable memory of the MI_CAM register is read, it contains the effective address and page sizes of an entry indexed by MI_CTR[ITLB_INDX]. MI_CAM is updated only by writing to it.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	EPN															
Reset	—															
R/W	R															
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/W															
SPR	816															

Figure 8-17. IMMU CAM Entry Read Register (MI_CAM)

Table 8-17 describes the MI_CAM fields.

Table 8-17. MI_CAM Field Descriptions

Bits	Name	Function
0–19	EPN	Effective page number
20–22	PS	Page size. (Values not shown are reserved) 000 4 Kbyte 001 16 Kbyte 011 512 Kbyte 111 8 Mbyte
23–26	ASID	Address space ID of the DTLB entry to be compared with M_CASID[CASID]
27	SH	Shared page 0 This entry matches only if the ASID field in the DTLB entry matches the value in M_CASID. 1 ASID comparison is disabled for the entry

Table 8-17. MI_CAM Field Descriptions (continued)

Bits	Name	Function
28	SPV	Subpage validity (subpage 0) 0 Subpage 0 (Address[20–21] = 00) is not valid 1 Subpage 0 (Address[20–21] = 00) is valid
29		0 Subpage 1 (Address[20–21] = 01) is not valid 1 Subpage 1 (Address[20–21] = 01) is valid
30		0 Subpage 2 (Address[20–21] = 10) is not valid 1 Subpage 2 (Address[20–21] = 10) is valid
31		0 Subpage 3 (Address[20–21] = 11) is not valid 1 Subpage 3 (Address[20–21] = 11) is valid

8.8.12.2 IMMU RAM Entry Read Register 0 (MI_RAM0)

The IMMU RAM entry read register 0 (MI_RAM0), shown in Figure 8-18, contains the physical page number and page attributes of an entry indexed by MI_CTR[ITLB_INDX]. This register is updated only when MI_CAM is updated.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	RPN															
Reset	—															
R/W	R															
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															
SPR	817															

Figure 8-18. IMMU RAM Entry Read Register 0 (MI_RAM0)

Table 8-18 describes MI_RAM0 fields.

Table 8-18. MI_RAM0 Field Descriptions

Bits	Name	Description
0–19	RPN	Real (physical) page number
20–22	PS_B	Page size. (Values not shown are reserved) 000 4 Kbyte 001 16 Kbyte 011 512 Kbyte 111 8 Mbyte
23	CI	Cache-inhibit attribute for the entry. 0 Caching is allowed. 1 Caching is inhibited.
24–27	APG	Access protection group. Up to 16 protection groups supported (uses one's complement format)

Table 8-18. MI_RAM0 Field Descriptions (continued)

Bits	Name	Description
28	SFP	Supervisor (supervisor) fetch permission 0 Subpage 0 (Address[20–21] = 00) Supervisor fetch is not permitted 1 Subpage 0 (Address[20–21] = 00) Supervisor fetch is permitted
29		0 Subpage 1 (Address[20–21] = 01) Supervisor fetch is not permitted 1 Subpage 1 (Address[20–21] = 01) Supervisor fetch is permitted
30		0 Subpage 2 (Address[20–21] = 10) Supervisor fetch is not permitted 1 Subpage 2 (Address[20–21] = 10) Supervisor fetch is permitted
31		0 Subpage 3 (Address[20–21] = 11) Supervisor fetch is not permitted 1 Subpage 3 (Address[20–21] = 11) Supervisor fetch is permitted

8.8.12.3 IMMU RAM Entry Read Register 1 (MI_RAM1)

The IMMU RAM entry read register 1 (MI_RAM1), shown in Figure 8-19, contains the protection mode information of the entry indexed by MI_CTR[ITLB_INDX]. This register is updated only when MI_CAM is written to.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Reset	0															
R/W	R															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—										UFP				PV	G
Reset	0										—				—	—
R/W	R															
SPR	818															

Figure 8-19. IMMU RAM Entry Read Register 1 (MI_RAM1)

Table 8-19 describes MI_RAM1 fields.

Table 8-19. MI_RAM1 Field Descriptions

Bits	Name	Description
0–25	—	Reserved

Table 8-19. MI_RAM1 Field Descriptions (continued)

Bits	Name	Description
26	UFP	User fetch permission 0 Subpage 0 (Address[20–21] = 00) User fetch is not permitted 1 Subpage 0 (Address[20–21] = 00) User fetch is permitted
27		0 Subpage 1 (Address[20–21] = 01) User fetch is not permitted 1 Subpage 1 (Address[20–21] = 01) User fetch is permitted
28		0 Subpage 2 (Address[20–21] = 10) User fetch is not permitted 1 Subpage 2 (Address[20–21] = 10) User fetch is permitted
29		0 Subpage 3 (Address[20–21] = 11) User fetch is not permitted 1 Subpage 3 (Address[20–21] = 11) User fetch is permitted
30	PV	Page validity 0 Page is not valid 1 Page is valid
31	G	Guarded memory attribute for entry 0 Nonguarded memory 1 Guarded memory

8.8.12.4 DMMU CAM Entry Read Register (MD_CAM)

When the DMMU CAM entry read register (MD_CAM), shown in Figure 8-20, is read, it contains the effective address and page sizes of an entry indexed by MD_CTR[DTLB_INDX]. This register is updated when a value is written to it.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	EPN															
Reset	—															
R/W	R(/W)															
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(/W)															
SPR	824															

Figure 8-20. DMMU CAM Entry Read Register (MD_CAM)

Table 8-20 describes MD_CAM fields.

Table 8-20. MD_CAM Field Descriptions

Bits	Name	Description
0–19	EPN	Effective page number

Table 8-20. MD_CAM Field Descriptions (continued)

Bits	Name	Description
20	SPVF	Subpage validity flags 0 Subpage 0 (address[20–21] = 00) is not valid 1 Subpage 0 (address[20–21] = 00) is valid
21		0 Subpage 1 (address[20–21] = 01) is not valid 1 Subpage 1 (address[20–21] = 01) is valid
22		0 Subpage 2 (address[20–21] = 10) is not valid 1 Subpage 2 (address[20–21] = 10) is valid
23		0 Subpage 3 (address[20–21] = 11) is not valid 1 Subpage 3 (address[20–21] = 11) is valid
24–26	PS	Page size. (Values not shown are reserved) 000 4 Kbyte 001 16 Kbyte 011 512 Kbyte 111 8 Mbyte
27	SH	Shared page 0 This entry matches only if the ASID field in the DTLB entry matches the value in M_CASID 1 ASID comparison is disabled for the entry
28–31	ASID	Address space ID of the DTLB entry to be compared with M_CASID[CASID]

8.8.12.5 DMMU RAM Entry Read Register 0 (MD_RAM0)

The DMMU RAM entry read register 0 (MD_RAM0), shown in Figure 8-21, contains the physical page number and page attributes of an entry indexed by MD_CTR[DTLB_INDX]. This register is updated when any value is written to MD_CAM.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	RPN															
Reset	—															
R/W	R															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	RPN				PS			APGI				G	WT	CI	—	
Reset	—															
R/W	R/W															
SPR	825															

Figure 8-21. DMMU RAM Entry Read Register 0 (MD_RAM0)

Table 8-21 describes MD_RAM0 fields.

Table 8-21. MD_RAM0 Field Descriptions

Bits	Name	Description
0–19	RPN	Real (physical) page number
20–22	PS	Page size. (Values not shown are reserved) 000 4 Kbyte 001 16 Kbyte 011 512 Kbyte 111 8 Mbyte
23–26	APGI	Access protection group inverted. Access protection group number in one's complement format
27	G	Guarded memory attribute for the entry 0 Nonguarded memory 1 Guarded memory
28	WT	Writethrough attribute for the entry 0 Copyback data cache policy page entry 1 Writethrough data cache policy page entry
29	CI	Cache-inhibit attribute for the entry. 0 Caching is allowed. 1 Caching is inhibited.
30–31	—	Reserved

8.8.13 DMMU RAM Entry Read Register 1 (MD_RAM1)

The DMMU RAM entry read register 1 (MD_RAM1), shown in Figure 8-22, contains the protection mode information of the entry indexed by MD_CTR[DTLB_INDX]. This register is updated only when a value is written to MD_CAM.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Reset	0															
R/W	R															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—	C	EVF	SA				SAT	URP0	UWP0	URP1	UWP1	URP2	UWP2	URP3	UWP3
Reset	0	—	—	—				—	—	—	—	—	—	—	—	—
R/W	R															
SPR	826															

Figure 8-22. DMMU RAM Entry Read Register 1 (MD_RAM1)

Table 8-22 describes MD_RAM1 fields.

Table 8-22. MD_RAM1 Field Descriptions

Bits	Name	Description
0–16	—	Reserved
17	C	Change bit for DTLB entry 0 Unchanged region. Write access to this page results in the implementation-specific IMMU exception invocation. Software should take an appropriate action before setting this bit to 1. 1 Changed region. Write access is allowed to this page.
18	EVF	Entry valid flag 0 Entry is invalid 1 Entry is valid
19	SA	Supervisor access 0 Subpage 0 (address[20–21] = 00) Supervisor access is not permitted 1 Subpage 0 (address[20–21] = 00) Supervisor access is permitted
20		0 Subpage 1 (address[20–21] = 01) Supervisor access is not permitted 1 Subpage 1 (address[20–21] = 01) Supervisor access is permitted
21		0 Subpage 2 (address[20–21] = 10) Supervisor access is not permitted 1 Subpage 2 (address[20–21] = 10) Supervisor access is permitted
22		0 Subpage 3 (address[20–21] = 11) Supervisor access is not permitted 1 Subpage 3 (address[20–21] = 11) Supervisor access is permitted
23	SAT	Supervisor access type 0 Supervisor access type is read only 1 Supervisor access type is read/write
24	URP0	User read permission page zero 0 Subpage 0 (address[20–21] = 00) User read access is not permitted 1 Subpage 0 (address[20–21] = 00) User read access is permitted
25	UWP0	User write permission page zero 0 Subpage 0 (address[20–21] = 00) User write access is not permitted 1 Subpage 0 (address[20–21] = 00) User write access is permitted
26	URP1	0 Subpage 1 (address[20–21] = 01) User read access is not permitted 1 Subpage 1 (address[20–21] = 01) User read access is permitted
27	UWP1	0 Subpage 1 (address[20–21] = 01) User write access is not permitted 1 Subpage 1 (address[20–21] = 01) User write access is permitted
28	URP2	0 Subpage 2 (address[20–21] = 10) User read access is not permitted 1 Subpage 2 (address[20–21] = 10) User read access is permitted
29	UWP2	0 Subpage 2 (address[20–21] = 10) User write access is not permitted 1 Subpage 2 (address[20–21] = 10) User write access is permitted
30	URP3	0 Subpage 3 (address[20–21] = 11) User read access is not permitted 1 Subpage 3 (address[20–21] = 11) User read access is permitted
31	UWP3	0 Subpage 3 (address[20–21] = 11) User write access is not permitted 1 Subpage 3 (address[20–21] = 11) User write access is permitted

8.9 Memory Management Unit Exceptions

Table 8-23 describes MPC855T-specific MMU exceptions.

Table 8-23. MPC855T-Specific MMU Exceptions

Exception	Cause
ITLB miss	MSR[IR] = 1 and an attempt is made to fetch an instruction from a page whose EPN cannot be translated by the ITLB. Tablewalk software is responsible for loading information for the missed page from the translation table. See Section 8.10.1.1, “Translation Reload Examples,” and Section 6.1.3.2, “Instruction TLB Miss Exception (0x01100).”
DTLB miss	MSR[DR] = 1 and an attempt is made to access a page whose EPN cannot be translated by the DTLB. Tablewalk software is responsible for loading translation information for the missed page from the translation table. See Section 8.10.1.1, “Translation Reload Examples,” and Section 6.1.3.3, “Data TLB Miss Exception (0x01200).”
ITLB error	The EA cannot be translated and the level-one segment or page valid bit is zero in the translation table, the fetch access violates memory protection, or the fetch access is to guarded memory and MSR[IR] = 1. The exact exception cause is found in SRR1. Table 6-15 describes bit assignments. If needed, it is software’s responsibility to invoke the ISI exception handler.
DTLB error	MSR[DR] = 1 and the EA of a load, store, **icbi**, **dcbz**, **dcbst**, **dcbf**, or **dcbi** cannot be translated and either the level-one segment or page valid bit are zero in the translation table, the access violates memory protection, or an attempt is made to write to a page with a negated change bit. The DSISR explains invocation of the DTLB error exception handler. Table 6-16 describes bit assignments. If needed, it is software’s responsibility to invoke the DSI exception handler.

8.10 TLB Manipulation

The TLBs can be updated in several ways. The TLB reloading process is primarily performed in software with some hardware assistance. The TLB replacement counter can be configured to select only from the first entries in each TLB. TLBs can be invalidated by using the **tlbie** and **tlbia** instructions.

8.10.1 TLB Reload

The TLB reload (tablewalk) function is performed in the software with some hardware assistance. It consists of the following actions:

- Automatic storage of the missed data or instruction EA and default attributes in MI_EPN or MD_EPN. This value is loaded into the selected entry on a write to MI_RPN or MD_RPN.
- Automatic updating of the replacement location counter to point to the entry to be replaced. This value is placed in the index field in MI_CTR and MD_CTR.
- As Figure 8-4 and Figure 8-5 show, the level-one pointer is generated when an **mfspr**[M_TWB] is performed by concatenating the level-one table base with the level-one index.

- The level-two pointer is generated when an **mfspr**[MD_TWC] is performed by concatenating the level-two table base (extracted from the level-one table) with the level-two index.
- The TLB entry is written by loading the tablewalk level-two entry value to M*x*_RPN.
- A scratch register, M_TW, is provided in addition to the architecture-defined SPRG0–SPRG3, so miss code need not corrupt existing GPRs.

8.10.1.1 Translation Reload Examples

The following examples reload a TLB entry using a two-level tree page table structure. In both examples, M_TWB holds the base pointer to the first-level table and data and instruction address translation are turned off. Figure 8-23 performs a DTLB reload.

```
dtlb_swtw   mtspr   M_TW, R1        # Save R1
            mfspr   R1, M_TWB       # Load R1 with level-1 pointer
            lwz     R1, (R1)        # Load level-1 page entry
            mtspr   MD_TWC,R1       # Save level-2 base pointer and level-1 attributes
            mfspr   R1, MD_TWC      # Load R1 with level-2 pointer while taking page
                                    # size into account
            lwz     R1, (R1)        # Load level-2 page entry
            mtspr   MD_RPN, R1      # Write TLB entry
            mfspr   R1, M_TW        # Restore R1
            rfi
```

Figure 8-23. DTLB Reload Code Example

Figure 8-24 performs an ITLB reload

```
itlb_swtw   mtspr   M_TW, R1        # Save R1
            mfspr   R1, SRR0        # Load R1 with instruction miss EA (the same data
                                    # may be taken from MI_EPN)
            mtspr   MD_EPN, R1      # Save instruction miss EA in MD_EPN
            mfspr   R1, M_TWB       # Load R1 with level-1 pointer
            lwz     R1, (R1)        # Load level-1 page entry
            mtspr   MI_TWC,R1       # Save level-1 attributes
            mtspr   MD_TWC,R1       # Save level-2 base pointer
            mfspr   R1, MD_TWC      # Load R1 with level-2 pointer while taking page
                                    # size into account
            lwz     R1, (R1)        # Load level-2 page entry
            mtspr   MI_RPN, R1      # Write TLB entry
            mfspr   R1, M_TW        # Restore R1
            rfi
```

Figure 8-24. ITLB Reload Code Example

8.10.2 Locking TLB Entries

Four entries in each TLB can be made unavailable to the replacement algorithm; thus by configuring the TLB replacement counters, the user can lock translation entries into them.

As shown in Figure 8-25, setting MI_CTR[RSV4I] or MD_CTR[RSV4D], configures the TLB replacement counter to select only from the first 28 entries in each TLB. Those fields also affect the **tlbia** instruction as described later. Replacement counters are cleared after a **tlbia** instruction executes. ITLB_INDX decrements after an ITLB reload; DTLB_INDX decrements after a DTLB reload.

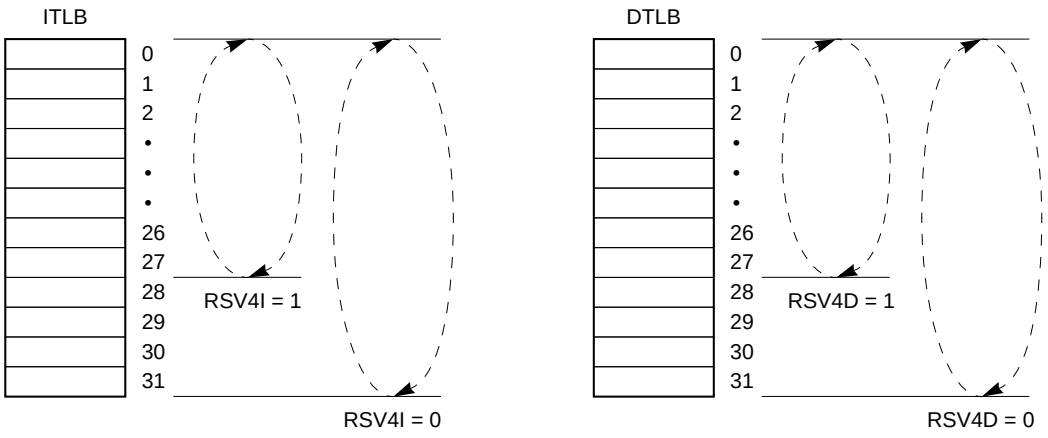


Figure 8-25. Configuring the TLB Replacement Counter

8.10.3 Loading Locked TLB Entries

The process of loading a single reserved entry in the TLB is as follows:

1. Disable the TLB by clearing MSR[IR] or MSR[DR] as needed.
2. Clear MI_CTR[RSV4I] (MD_CTR[RSV4D]).
3. Invalidate the EA of the reserved page by using **tlbia** or **tlbie**.
4. Set MI_CTR[ITLB_INDX] (MD_CTR[DTLB_INDX]) to the appropriate value (between 27 and 31).
5. Load M*x*_EPN with the effective page number, the ASID of the reserved page, and set EV.
6. Run software tablewalk code to load the appropriate entry into the translation lookaside buffer. See Section 8.10.1.1, “Translation Reload Examples.”
7. Repeat steps 4–6 to load other TLB entries.
8. Set MI_CTR[RSV4I] (MD_CTR[RSV4D]).

8.10.4 TLB Invalidation

Executing **tlbie** invalidates TLB entries that hit, including reserved entries. Note that EA[0–21] is used in the comparison because segment registers as defined by the PowerPC architecture are not implemented. Although for entries with pages larger than 4 Kbytes, some lower bits of the effective page number are ignored. The ASID value in the entry is ignored for the purpose of matching an invalidate address, thus multiple entries can be invalidated if they have the same effective address and different ASID values.

Executing **tlbia** invalidates all entries in both TLBs, however if MI_CTR[RSVI] or MD_CTR[RSVD] is set, the reserved entries are not invalidated. Software can explicitly invalidate one or more of these entries by setting MD_CTR[DTLB_INDX] or MI_CTR[ITLB_INDX], negating MD_EPN[EV] or MI_EPN[EV], and writing to the appropriate MD_RPN or MI_RPN. The TLBs are not invalidated automatically on reset, but are disabled. However, they must be invalidated under program control during initialization.

Chapter 9
Instruction Execution Timing

This chapter describes the timing of instructions that execute in the core. Examples show stalls and bubbles due to serialization, latency, and blockage.

9.1 Instruction Execution Timing Examples

The following sections provide timings for the following scenarios:

- Data cache load
- Writeback arbitration
- Private writeback bus load
- Fastest external load (data cache miss)
- Full completion queue (CQ)
- Branch instruction handling
- Branch prediction

All examples assume an instruction cache hit.

9.1.1 Data Cache Load with a Data Dependency

Figure 9-1 shows a data cache load with zero wait states. The **sub** instruction depends on the value loaded to **r**12,which causes a bubble in the instruction stream. The example in Section 9.1.3, “Private Writeback Bus Load,” has no such dependency.

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

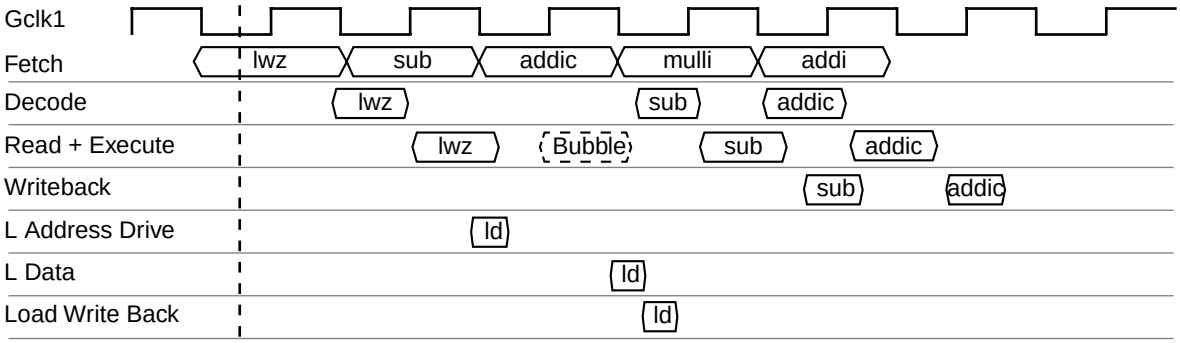


Figure 9-1. Data Cache Load Timing

9.1.2 Writeback Arbitration

In Figure 9-2, the **addic** instruction is dependent on the **mulli** result. Because the single-cycle instruction **sub** has priority on the writeback bus over the **mulli**, the **mulli** writeback is delayed one clock and causes a bubble in the execute stream.

```
mulli   r12,r4,3
sub     r3,r15,3
addic   4,r12,1
```

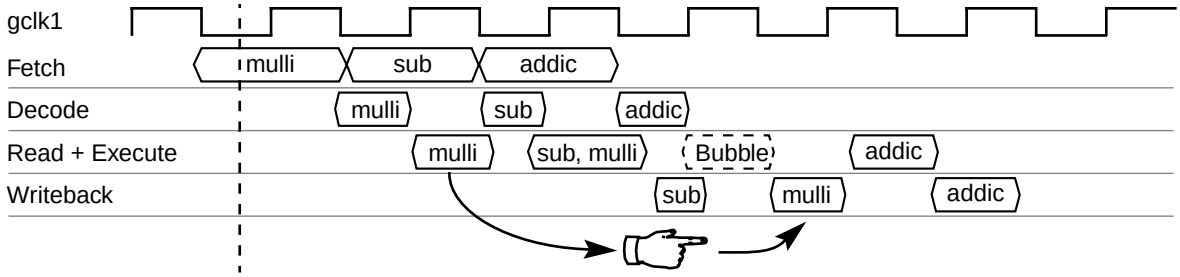


Figure 9-2. Writeback Arbitration Timing—Example 1

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

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

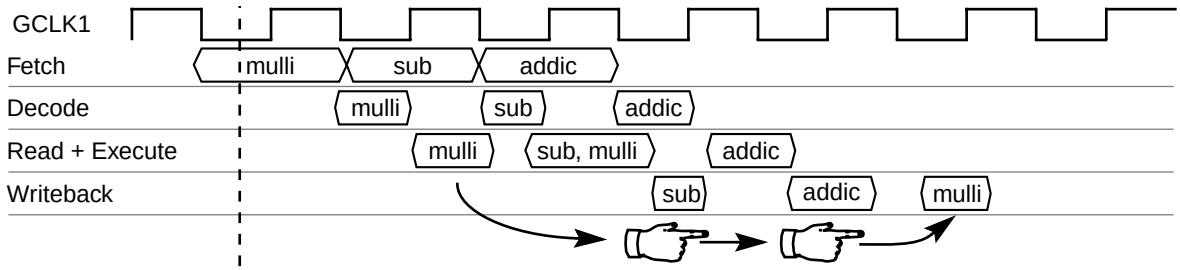


Figure 9-3. Writeback Arbitration Timing—Example 2

9.1.3 Private Writeback Bus Load

In Figure 9-4, **lwz** and **xor** write back in the same clock since they use the writeback bus in two different ticks (a tick = 1/4 of a processor clock).

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

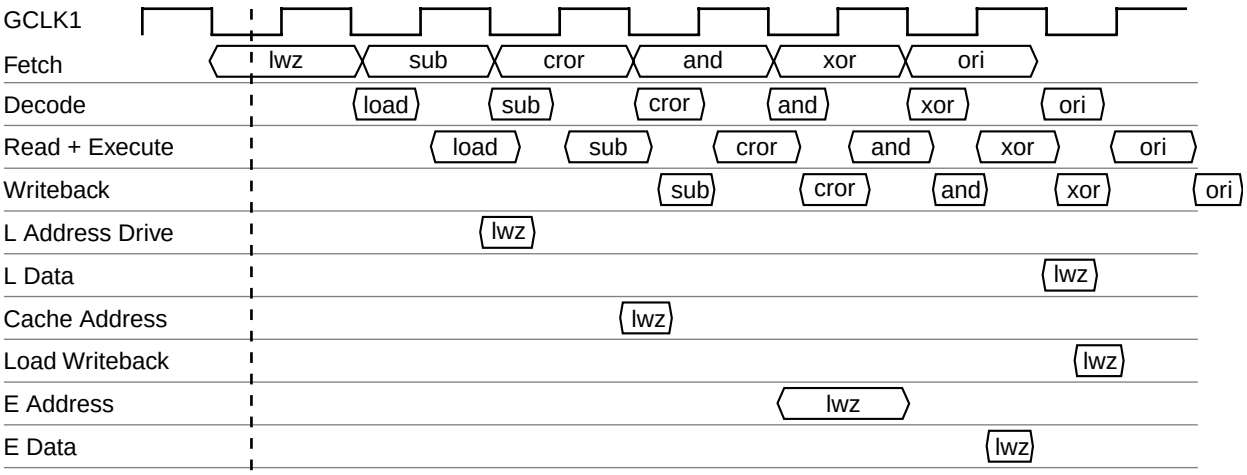


Figure 9-4. Private Writeback Bus Load Timing

9.1.4 Fastest External Load (Data Cache Miss)

Figure 9-5 shows a **sub** instruction dependent on the value read by the load. It causes three bubbles in the execution stream. Assuming SCCR[EBDF] = 00, the external clock (CLKOUT) is shifted 90° from the internal clock (GCLK1).

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

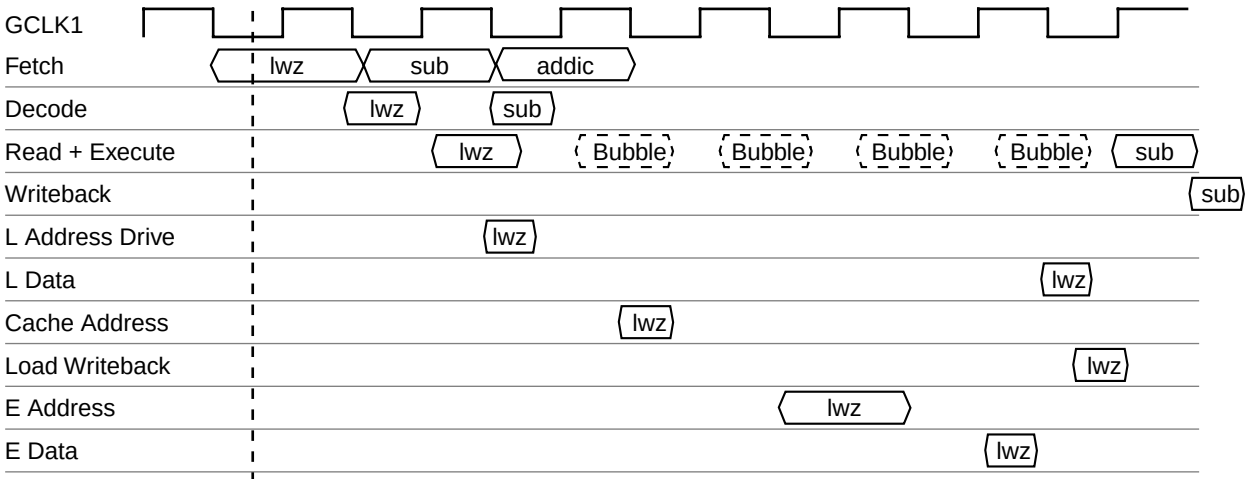


Figure 9-5. External Load Timing

9.1.5 A Full Completion Queue

Figure 9-6 shows stalls due to a full CQ. Here, the CQ is full from executing **sub**, **addic**, and **and**. It takes one more bubble from the load writeback to allow further issue while the CQ retires **sub**, **addic**, and **and**.

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

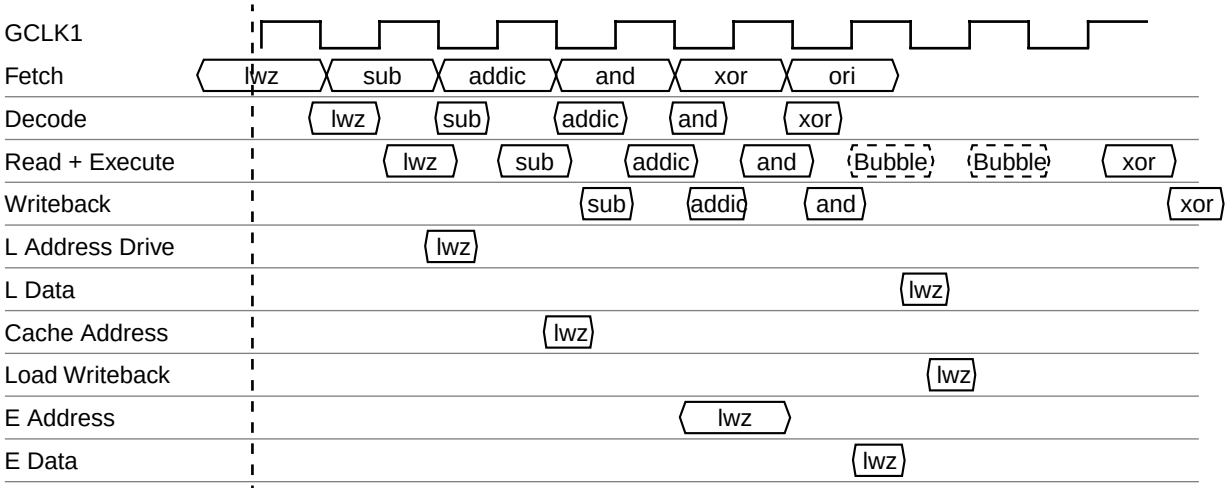


Figure 9-6. Full Completion Queue Timing

9.1.6 Branch Instruction Handling

In Figure 9-7 the **lwz** instruction accesses internal memory with one wait state. The IQ and parallel operation of the BPU allows the two bubbles caused by the **bl** issue and execution to overlap the two bubbles caused by the load. Issuing **bl** causes a bubble because it does no work.

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

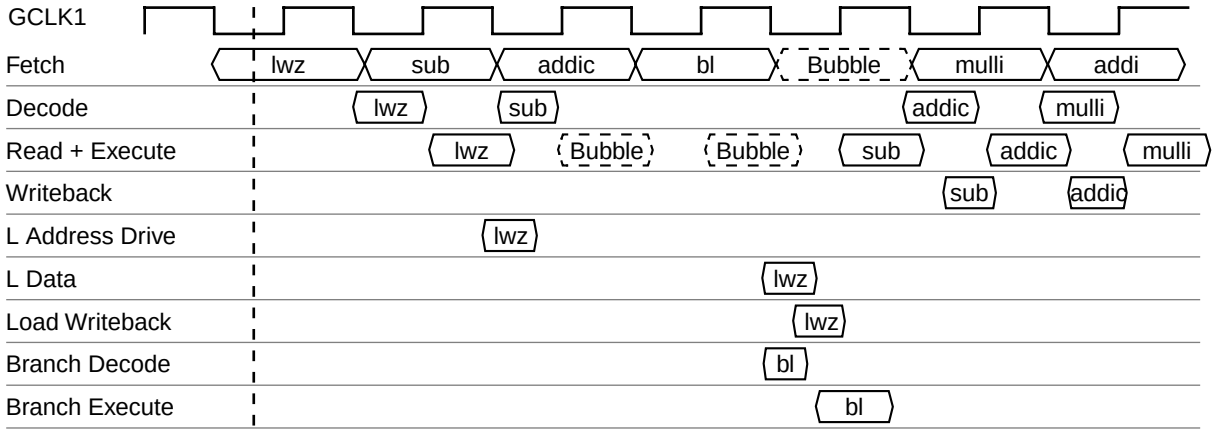


Figure 9-7. Branch Folding Timing

9.1.7 Branch Prediction

In this example, the **blt** instruction is dependent on the **cmpi** instruction. Nevertheless, the BPU predicts the correct path and allows an overlap of its bubbles with those of **lwz**. When **cmpi** writes back, the BPU reevaluates the decision. If the prediction is correct, no more action is taken and execution continues. Instructions on the predicted path cannot be dispatched before the condition is resolved.

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

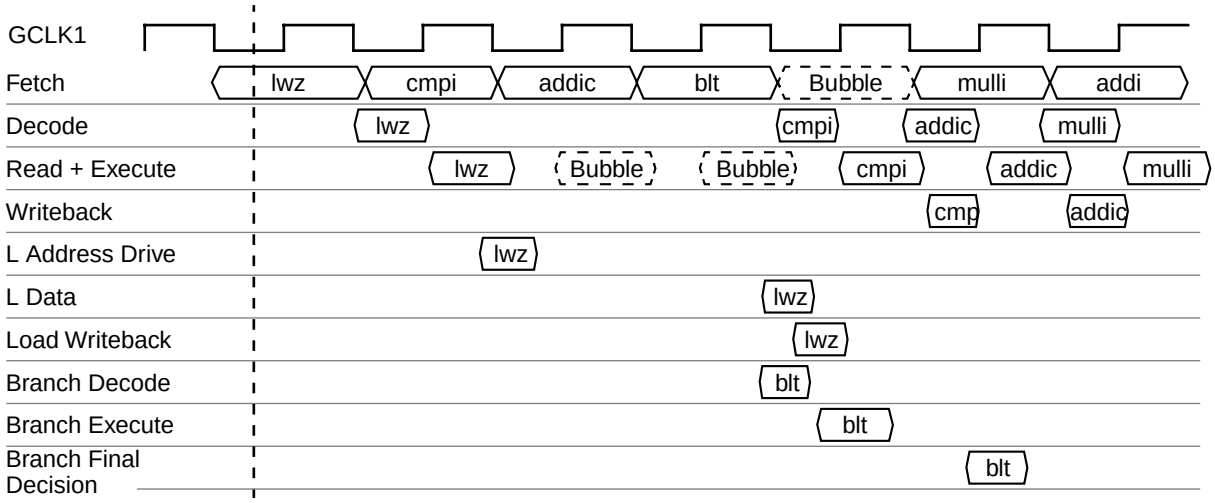


Figure 9-8. Branch Prediction Timing

9.2 Instruction Timing List

Table 9-1 summarizes instruction execution timings in terms of latency and blockage of the appropriate execution unit. A serializing instruction blocks all execution units.

Table 9-1. Instruction Execution Timing

Instructions	Latency	Blockage	Unit	Serializing
Branch: **b**, **ba**, **bl**, **bla**, **bc**, **bca**, **bcl**, **bcla**, **bclr**, **bclrl**, **bcctr**, **bcctl**	Taken 2	2	BPU	No
	Not taken 1	1		
System call: **sc**, **rfi**	Serialize + 2		—	Yes
CR logical: **crand**, **crxor**, **cror**, **crnand**, **crnor**, **crandc**, **creqv**, **crorc**, **mcrf**	1	1	BPU	No
Integer trap: **twi**, **tw**	Taken serialize + 3	Serialize + 3	IU	After
	Not taken 1	1		No
Move to: **mtspr**, **mtcrf**, **mtmsr**, **mcrxr** except **mtspr** to LR and CTR and to SPRs external to the core.	Serialize + 1		All	Yes
Move to LR, CTR: **mtspr**	1	1	BPU	No
Move to SPRs external to core: **mtspr**, **mttb**, **mttbu**. See Section 9.2.3, “Accessing Off-Core SPRs.”	Serialize + 1 [1]	Serialize + 1	LSU	Yes
Move from SPRs external to core: **mfspr**, **mftb**, **mftbu**	Load latency	1	LSU	No
Move from SPRs internal to core: **mfspr** [2]	1		—	See list [3]
Move from: **mfcr**, **mfmsr**	Serialize + 1		—	See list [4]
Integer arithmetic: **addi**, **add**, **addis**, **subf**, **addic**, **subfic**, **addic.**, **addc**, **adde**, **subfc**, **subfe**, **addme**, **addze**, **subfme**, **subfze**, **neg**	1		IU	No

Table 9-1. Instruction Execution Timing (continued)

Instructions	Latency	Blockage	Unit	Serializing
Integer divide: **divw**, **divwu**	Min 2 Max 11 [5]	Min 2 Max 11 [6]	IU	No
Integer multiply: **mul**, **mullw**, **mulhw**, **mulhwu**	2	1-2 [7]	IU	No
Integer compare: **cmpi**, **cmp**, **cmpli**, **cmpl**	1		IU	No
Integer logical: **andi.**, **andis.**, **ori**, **oris**, **xori**, **xoris**, **and**, **or**, **xor**, **nand**, **nor**, **eqv**, **andc**, **orc**, **extsb**, **extsh**, **cntlzw**	1		IU	No
Integer rotate and shift: **rlwinm**, **rlwnm**, **rlwimi**, **slw**,**srw**, **srawi**, **sraw**	1		IU	No
Integer load: **lbz**, **lbzu**, **lbzx**, **lbzux**, **lhz**, **lhzu**, **lhzx**, **lhzux**, **lha**, **lhau**, **lhax**, **lhaux**, **lwz**, **lwzu**, **lwzx**, **lwzux**, **lhbrx**, **lwbrx.**	2 [8]	1	LSU	No
Integer store: **stb**, **stbu**, **stbx**, **stbux**, **sth**, **sthu**, **sthx**, **sthux**, **stw**, **stwu**, **stwbrx**, **stwx**, **stwux**, **sthbrx**	1[1]	1	LSU	No
Integer load/store multiple: **lmw**, **smw**	Serialize + 1 + no. of registers		LSU	Yes
Synchronize: **sync**	Serialize + 1		LSU	Yes
Memory synchronization: **lwarx**, **stwcx.**	Serialize + 2		LSU	Yes
Move CR from XER: **mcrxr**	Serialize + 1		LSU	Yes
Move to/from SPR (Debug, DAR, DSISR): **mtspr**, **mfspr**	Serialize + 1		LSU	Yes
String instructions: **lswi**, **lswx**, **stswi**, **stswx**. See Section 9.2.2, “String Instruction Latency.”	Serialize + 1 + no. of words accessed		LSU	Yes
Memory control instructions: **isync**	Serialize		BPU	Yes
Order memory access: **eieio**	1		LSU	Next load/store is synchronized with ones before
Cache control: **icbi**	1		LSU, I-cache	No

1 Although a store (as well as **mtspr** for SPRs external to the core) issued to the LSU buffer frees the core pipeline, the next load or store is not performed on the bus until it is free.

2 See Table 4-5.

3 Refer to Chapter 4, “MPC8xx Core Register Set.”

4 See Section 4.1.1.1.1, “Condition Register (CR),” and Section 4.1.2.3.1, “Machine State Register (MSR).”

5 Where

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

$$DivisionLatency = \begin{bmatrix} NoOverflow \Rightarrow 3 + \left(\frac{34 - divisorLength}{4}\right) \\ Overflow \Rightarrow 2 \end{bmatrix}$$

6 Division blockage = division latency.

7 Blockage of the multiply instruction is dependent on the next instruction.If the next instruction is a divide, the blockage is 2 clocks; otherwise, the blockage is 1 clock.

8 Assumes nonspeculative aligned access, on-chip memory, and available bus. See Section 3.6.3.4, “Nonspeculative Load Instructions,” Section 3.6.3.5, “Unaligned Accesses,” and Section 9.2.1, “Load/Store Instruction Timing.”

9.2.1 Load/Store Instruction Timing

Table 9-2. summarizes load/store instruction timings. This table assumes zero wait-state memory references on a parked bus and pipelined external memory accesses.

Table 9-2. Load/Store Instructions Timing

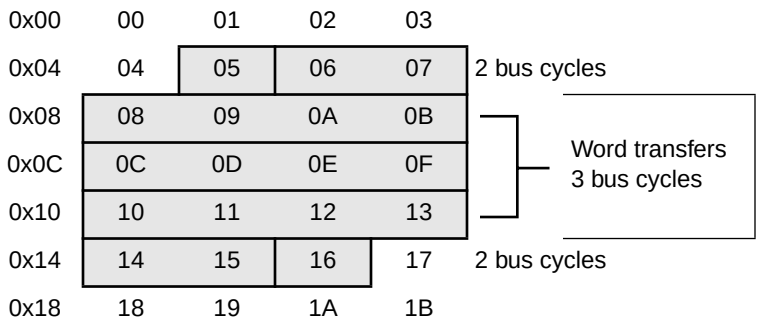
Instruction Type	Latency		Cleared from LSU	
	Data Cache	External Memory	Data Cache	External Memory
Integer single target register load (aligned)	2 cycles	5 cycles	2 cycles	5 cycles
Integer single target register store (aligned)	1 cycle	1 cycle	2 cycles	5 cycles
Load/store multiple	1 + N [1]	$3 + N + \left(\frac{N+1}{3}\right)$	1 + N	$3 + N + \left(\frac{N+1}{3}\right)$

1 N denotes the number of registers transferred.

9.2.2 String Instruction Latency

String accesses require separate aligned bus accesses. Figure 9-9 shows the maximum number of bus cycles needed for string accesses where the beginning and end are unaligned.

Figure 9-9. Bus Latency for String Instructions



9.2.3 Accessing Off-Core SPRs

The LSU handles **mtspr** and **mfspr** accesses to off-core SPRs by using a special cycle on the internal bus. See Section 4.1.3.1, “Accessing SPRs.” If the access ends in a bus error, a software emulation exception is taken. All write operations to off-core SPRs (**mtspr**) are previously synchronized. In other words, the instruction is not taken until all prior instructions terminate.

Part III
Configuration and Reset

Audience

Part III is intended for system designers and programmers who need to understand the operation of the MPC855T at start up. It assumes an understanding of the programming model described in the previous chapters and a high level understanding of the MPC855T.

Contents

Part III describes start-up behavior of the MPC855T. It contains the following chapters:

- Chapter 10, “System Interface Unit,” describes the SIU, which controls system start-up, initialization and operation, protection, as well as the external system bus.
- Chapter 11, “Reset,” describes the behavior of the MPC855T at reset and start-up.

Suggested Reading

Supporting documentation such as technical specifications, reference materials, and detailed applications notes can be accessed through the world-wide web at http://www.motorola.com

Conventions

This chapter uses the following notational conventions:

Bold	Bold entries in figures and tables showing registers and parameter RAM should be initialized by the user.
mnemonics	Instruction mnemonics are shown in lowercase bold.
italics	Italics indicate variable command parameters, for example, **bcctr***x*. Book titles in text are set in italics.
0x0	Prefix to denote hexadecimal number

0b0	Prefix to denote binary number
rA, **r**B	Instruction syntax used to identify a source GPR
rD	Instruction syntax used to identify a destination GPR
REG[FIELD]	Abbreviations or acronyms for registers or buffer descriptors are shown in uppercase text. Specific bits, fields, or numerical ranges appear in brackets. For example, MSR[LE] refers to the little-endian mode enable bit in the machine state register.
x	In certain contexts, such as in a signal encoding or a bit field, indicates a don’t care.
n	Indicates an undefined numerical value

Acronyms and Abbreviations

Table i contains acronyms and abbreviations that are used in this document. Note that the meanings for some acronyms (such as SDR1 and DSISR) are historical, and the words for which an acronym stands may not be intuitively obvious.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
BIST	Built-in self test
CRC	Cyclic redundancy check
CTR	Count register
DABR	Data address breakpoint register
DAR	Data address register
DEC	Decrementer register
DMA	Direct memory access
DRAM	Dynamic random access memory
DTLB	Data translation lookaside buffer
EA	Effective address
GPR	General-purpose register
IEEE	Institute of Electrical and Electronics Engineers
ITLB	Instruction translation lookaside buffer
LSB	Least-significant byte
lsb	Least-significant bit
LSU	Load/store unit
MMU	Memory management unit
MSB	Most-significant byte

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
msb	Most-significant bit
MSR	Machine state register
PCI	Peripheral component interconnect
RISC	Reduced instruction set computing
RTOS	Real-time operating system
Rx	Receive
SPR	Special-purpose register
SWT	Software watchdog timer
TB	Time base register
TLB	Translation lookaside buffer
Tx	Transmit

Chapter 10
System Interface Unit

The system interface unit (SIU) 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 (PIT), decrementer, timebase, and real-time clock. The clock synthesizer generates the clock signals for other modules and external devices that the SIU interfaces with. The SIU 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; the baud rate generators and serial ports work with a fixed frequency. For more information, see Chapter 14, “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 MPC855T is designed to allow external bus devices to request and obtain system bus mastership. Chapter 12, “External Signals,” describes bus operation. The memory controller module provides a glueless interface to many types of memory devices and peripherals; it supports a maximum of eight memory banks, each with its own device and timing attributes. Memory control services are provided to both internal and external masters. The MPC855T supports circuit board test strategies through user-accessible test logic that is fully compliant with the IEEE 1149.1 standard described in Chapter 45, “IEEE 1149.1 Test Access Port.”

The PCMCIA host adapter module provides all control logic for a PCMCIA interface. This interface complies fully with the PCMCIA standard, Release 2.1+ (PC Card -16). It can support PCMCIA socket with a maximum of eight memory or I/O windows.

10.1 Features

The following is a list of the SIU’s main features:

- System configuration and protection
- System interrupt configuration
- System reset monitoring and generation
- Clock synthesizer

- Power management
- Real-time clock
- Decrementer
- Time base
- Periodic interrupt timer (PIT)
- External bus interface control
- Eight memory banks supported by the memory controller
- Debug support
- PCMCIA host adapter module supports slot with eight memory or I/O windows
- IEEE 1149.1 test access port

10.2 System Configuration and Protection

The MPC855T incorporates many system functions that normally must be provided in external circuits. The following features provide maximum system safeguards against hardware and/or software faults:

- System configuration—Allows control of parity checking, show cycle operation, and part and mask number constants.
- Bus monitor—Monitors the $\overline{\text{TA}}$ response time for bus accesses initiated by internal masters. $\overline{\text{TEA}}$ is asserted if the $\overline{\text{TA}}$ response limit is exceeded. The bus monitor measures time between $\overline{\text{TS}}$ and any termination of the bus cycle, including $\overline{\text{TA}}$, $\overline{\text{TEA}}$, and $\overline{\text{RETRY}}$.
- Software watchdog timer (SWT)—Asserts a reset or nonmaskable interrupt (NMI) that is selected by the system protection control register (SYPCR) if software fails to service this timer after a certain period. After system reset, the timer, if enabled, selects a maximum time-out period and asserts $\overline{\text{SRESET}}$ or NMI (system reset interrupt) if the time-out is reached. This timer can be disabled or its time-out period can be changed in SYPCR. Once SYPCR is written, it cannot be written again until a system reset.
- Periodic interrupt timer (PIT)—Generates periodic interrupts for use with a real-time operating system (RTOS) or the application software. The PIT is clocked by the PITRTCLK clock, thus providing a period from 122 µs to 8 seconds assuming a 32.768-KHz crystal. The PIT can be disabled if it is not needed.
- 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 and control register (TBSCR) is set for the reference register that generated the interrupt. The timebase is clocked by the TMBCLK clock.

- Decrementer—Provides a decrementer register/interrupt clocked at the timebase frequency. This 32-bit decrementing counter is defined to be clocked by TMBCLK. When it is driven by a 4-MHz oscillator the period for the decrementer is 4,295 seconds (approximately 71.6 minutes).
- Real-time clock (RTC)—Provides 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 register. The RTC is clocked by PITRTCLK.
- Freeze support—The SIU determines whether the software watchdog timer, PIT, timebase, decrementer, and real-time clock should continue to run in freeze mode.

Figure 10-1 is a block diagram of the system configuration and protection logic.

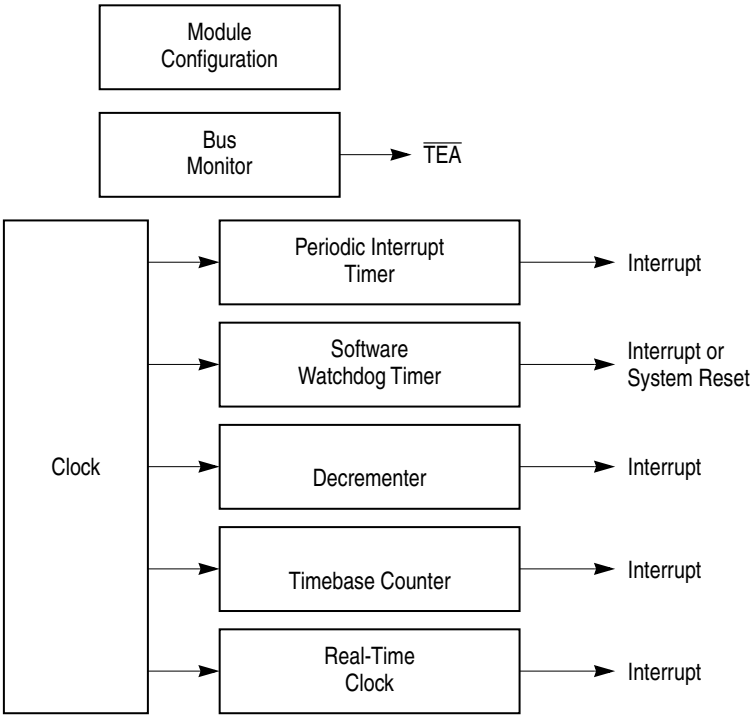


Figure 10-1. System Configuration and Protection Logic

10.3 Multiplexing SIU Pins

Due to the limited number of pins available in the MPC855T package, some of the functionalities share pins. Table 10-1 shows how functionality is controlled on each pin.

Table 10-1. Multiplexing Control

Name	Pin Configuration Control
TSIZ0/REG	Dynamically active 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 SIUMCR.
CS6/CE1_B CS7/CE2_B	Address matching and bank valid bits. When a transfer matches either memory controller bank 6 or any PCMCIA bank mapped to slot B, CS6/CE1_B is asserted. When a transfer matches either memory controller bank 7 or any PCMCIA bank mapped to slot B, CS7/CE2_B is 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. See Section 11.3.1.1, “Hard Reset Configuration Word.”
OP2/MODCK1/STS OP3/MODCK2/DSDO	At power-on reset, this functions as MODCK[1–2]. Otherwise, programmed in the SIUMCR and hard reset configuration.

10.4 Programming the SIU

The following sections describe registers used for programming the SIU.

10.4.1 Internal Memory Map Register (IMMR)

The internal memory map register (IMMR) is an SPR that identifies specific devices and the internal memory map base address. Using **mfspr**, software can read IMMR to

determine the location and availability of any on-chip system resource. ISB can be written by **mtspr**, but PARTNUM and MASKNUM are mask programmed and cannot be changed.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	ISB															
Reset	Set by reset configuration															
R/W	R/W															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	PARTNUM								MASKNUM							
Reset	Value depends on the part revision								Value depends on the mask revision							
R/W	R								R							
SPR	638															

Figure 10-2. Internal Memory Map Register (IMMR)

Table 10-2 describes IMMR fields.

Table 10-2. MMR Field Descriptions

Bits	Name	Description
0–15	ISB	Internal space base. Defines the base address of the internal memory space. At reset, ISB can be configured to one of four addresses and changed to any value by the software. The number of programmable ISB bits and the resolution of the location of internal space depends on the implementation's internal memory space. In the MPC855T, all 16 bits can be programmed. Chapter 2, "Memory Map," describes the internal memory map. Section 11.3.1.1, "Hard Reset Configuration Word," describes available and default initial values.
16–23	PARTNUM	Part number (read-only). Mask programmed with a code corresponding to the part number of the MPC855T. Intended to help factory test and user code that is sensitive to part refinements. PARTNUM 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 MPC855T's part number can be found at www.motorola.com, on itsproduct summary page.
24–31	MASKNUM	Mask number. (read-only) Mask programmed with a code corresponding to the mask number of the MPC855T. Intended to help factory test and user code that is sensitive to part refinements. The MPC855T's mask number can be found at www.motorola.com, on itsproduct summary page

For the latest documentation on part/revision numbers and microcode REV_NUMs, see the website at www.motorola.com.

10.4.2 SIU Module Configuration Register (SIUMCR)

The SIU module configuration register (SIUMCR) contains bits that configure the following features in the SIU:

- External bus arbitration
- External master support
- Debug and test port configuration

- System interface pin configuration
- Parity support

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	EARB	EARP			—				DSHW	DBGC		DBPC		—	FRC	DLK
Reset	*n*	000_0001_0								*n*		*n*		000		
R/W	R/W															
Addr	(IMMR & 0xFFFF0000) + 0x000															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	OPAR	PNCS	DPC	MPRE	MLRC		AEME	SEME		GB5E	B2DD	B3DD	—			
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	(IMMR & 0xFFFF0000) + 0x002															

Figure 10-3. SIU Module Configuration Register (SIUMCR)

This register is affected by $\overline{\text{HRESET}}$ but is not affected by $\overline{\text{SRESET}}$. Table 10-3 describes SIUMCR fields.

Table 10-3. SIUMCR Field Descriptions

Bits	Name	Description
0	EARB	External arbitration. For more information, see Section 13.4.6, “Arbitration Phase.” The default value depends on the reset configuration; see Section 11.3.1.1, “Hard Reset Configuration Word.” 0 Internal arbitration is performed. 1 External arbitration is assumed.
1–3	EARP	External arbitration request priority. Defines the priority of the external master’s arbitration request relative to requests by internal modules. Valid when EARB is cleared. 000 = lowest priority and 111 = highest (however, the internal UPM-based refresh cycles always have a higher priority and will preempt any external master if the internal arbiter is used). See Figure 13-21 and Table 19-1.
4–7	—	Reserved, should be cleared.
8	DSHW	Data show cycles. Selects the show cycle mode to be applied to data cycles. Data show cycles do not include CPU interaction with the data cache; they only include CPU interactions with peripherals on the internal U-bus (that is, CPM and SIU). (Instruction show cycles are programmed in ICTRL see the *Hardware Specifications* 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.
9–10	DBGC	Debug pin configuration. The default is set by the hard reset configuration word. See Section 11.3.1.1, “Hard Reset Configuration Word” for the description of these bits.
11–12	DBPC	Debug port pins configuration. Determines the active pins for the development port. The default is set by the hard reset configuration word. See Section 11.3.1.1, “Hard Reset Configuration Word” for the description of these bits.
13	—	Reserved, should be cleared.
14	FRC	FRZ pin configuration. Configures the functionality of FRZ/$\overline{\text{IRQ6}}$. 0 FRZ/$\overline{\text{IRQ6}}$ functions as FRZ. 1 FRZ/$\overline{\text{IRQ6}}$ functions as $\overline{\text{IRQ6}}$.

Table 10-3. SIUMCR Field Descriptions (continued)

Bits	Name	Description
15	DLK	Debug register lock. If DLK is set, bits 8–15 are locked and writes to those bits are no longer performed. These bits can be written once the internal FRZ signal is asserted, regardless of the state of DLK. Cleared at reset.
16	OPAR	Odd parity. Used to program odd or even parity. Also used to generate parity errors for testing purposes by writing the memory with OPAR = 1 and reading the memory with OPAR = 0.
17	PNCS	Parity enable for nonmemory controller regions. Enables parity generation/checking for memory regions not controlled by the memory controller.
18	DPC	Data parity pins configuration. Configures the functionality of DP[0–3]/IRQ[3–6]. 0 DP[0–3]/IRQ[3–6] functions as IRQ[3–6]. 1 DP[0–3]/IRQ[3–6] functions as DP[0–3].
20–21	MLRC	Multi-level reservation control. Configures the functionality of KR/RETRY/IRQ4/SPKROUT. 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.
22	AEME	Asynchronous external master enable. Configures how the memory controller refers to external asynchronous masters initiating a transaction. If AEME = 1, the memory controller interprets any assertion of AS as an external asynchronous master initiating a transaction. If it is reset, the memory controller ignores the value of AS.
23	SEME	Synchronous external master enable. Configures how the memory controller refers to synchronous external masters initiating a transaction. If SEME = 1, the memory controller interprets any assertion of TS not driven by the MPC855T as a synchronous external master initiating a transaction. If SEME = 0, the memory controller ignores TS unless it is external bus master.
24	BSC	Configures how memory controller and PCMCIA interface byte selects and strobes are configured. 0 BS_A[0–3] are driven just on their dedicated pins. WE0/BS_B0/IORD is driven on its dedicated pin. WE1/BS_B1/IOWR is driven on its dedicated pin. WE2/BS_B2/PCOE is driven on its dedicated pin. WE3/BS_B3/PCWE is driven on its dedicated pin. 1 Assertion of either BS_A0, WE0, BS_B0 or IORD is driven on BS_A0 and WE0/BS_B0/IORD. Assertion of either BS_A1, WE1, BS_B1 or IOWR is driven on BS_A1 and WE1/BS_B1/IOWR. Assertion of either BS_A2, WE2, BS_B2 or PCOE is driven on BS_A2 and WE2/BS_B2/PCOE. Assertion of either BS_A3, WE3, BS_B3 or PCWE is driven on BS_A3 and WE3/BS_B3/PCWE.
25	GB5E	GPL_B5 enable 0 The BDIP functionality is active. 1 The GPL_B5 of the memory controller functionality is active
26	B2DD	Bank 2 double drive. If this bit is set, CS2 is reflected on GPL_x2.
27	B3DD	Bank 3 double drive. If this bit is set, CS3 is reflected on GPL_x3.
28–31	—	Reserved, should be cleared.

10.4.3 System Protection Control Register (SYPCR)

The system protection control register (SYPCR) controls the system monitors and bus monitor timing. It can be read at any time, but can be written only once after system reset. This register is affected by HRESET but is not affected by SRESET.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	SWTC															
Reset	1111_1111_1111_1111															
R/W	R/W															
SPR	(IMMR & 0xFFFF0000) + 0x004															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	BMT								BME	—			SWF	SWE	SWRI	SWP
Reset	1111_1111								0	000			0	1	1	1
R/W	R/W															
SPR	(IMMR & 0xFFFF0000) + 0x006															

Figure 10-4. System Protection Control Register (SYPCR)

Table 10-4 describes SYPCR fields.

Table 10-4. SYPCR Field Descriptions

Bits	Name	Description
0–15	SWTC	Software watchdog timer count. Count value for the software watchdog timer.
16–23	BMT	Bus monitor timing. Defines the timeout period, in 8 system clock resolution, for the bus monitor. maximum timeout is 2,040 clocks.
24	BME	Bus monitor enable. Controls bus monitor operation during internal-to-external bus cycles. 0 Disable the bus monitor. 1 Enable the bus monitor. Note: If the bus monitor is disabled, transfer error conditions do not cause $\overline{\text{TEA}}$ to be asserted.
25–27	—	Reserved, should be cleared.
28	SWF	Software watchdog freeze 0 The software watchdog timer continues counting even if FRZ is asserted. 1 The software watchdog timer stops counting when FRZ is asserted.
29	SWE	Software watchdog enable. To disable the software watchdog timer, it should be cleared by the software after a system reset. 0 Software watchdog timer disabled. 1 Software watchdog timer enabled. (default)
30	SWRI	Software watchdog reset/interrupt select. 0 The software watchdog timer causes an NMI (system reset interrupt) to the core. 1 The software watchdog timer causes an $\overline{\text{HRESET}}$. (default)
31	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. (default)

10.4.4 Transfer Error Status Register (TESR)

The transfer error status register (TESR) has a bit for each transfer error exception source. Set bits indicate what type of transfer error exception that occurred since bits were last

cleared. Bits are cleared by reset or by writing ones to them. Canceled speculative accesses that do not cause an interrupt may set these bits. TESR has two identical sets of fields, one for instruction transfers and one for data transfers.This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Reset	xxxx_xxxx_xxxx_xxxx															
R/W	R/W															
SPR	(IMMR & 0xFFFF0000) + 0x020															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—		IEXT	ITMT	IPB0	IPB1	IPB2	IPB3	—		DEXT	DTMT	DPB0	DPB1	DPB2	DPB3
Reset	0000_0000_0000_0000															
R/W	R/W															
SPR	(IMMR & 0xFFFF0000) + 0x022															

Figure 10-5. Transfer Error Status Register (TESR)

Table 10-5 describes TESR fields.

Table 10-5. TESR Field Descriptions

Bits	Name	Description
0–17	—	Reserved, should be cleared.
18	IEXT	Instruction external transfer error acknowledge. Set if the cycle is terminated by an externally generated $\overline{\text{TEA}}$ when an instruction fetch is initiated.
19	ITMT	Instruction transfer monitor timeout. Set if the cycle is terminated by a bus monitor timeout when an instruction fetch is initiated.
20–23	IPB[0–3]	Instruction parity error on bytes 0–3. Each byte lane has four parity error status bits; one is set for the byte that had a parity error when an instruction was fetched. Parity check for memory not controlled by the memory controller is enabled by SIUMCR[PNCS], see Table 10-3.
24–25	—	Reserved, should be cleared.
26	DEXT	Data external transfer error acknowledge. 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.
27	DTMT	Data transfer monitor timeout. Set if the cycle is terminated by a bus monitor timeout when a data load or store is requested by an internal master.
28–31	DPB[0–3]	Data parity error on bytes 0–3. Each byte lane has four parity error status bits; one is set for the byte that had a parity error when an internal master requested a data load. Parity checking for memory not controlled by the memory controller is enabled by SIUMCR[PNCS], see Table 10-3.

10.4.5 Register Lock Mechanism

If the MPC855T sets PLPRCR[LPM] = 11 before entering power-down mode, then the registers of the SIU maintained by KAPWR are automatically protected. However, to

provide protection of the SIU registers maintained by KAPWR against uncontrolled shutdown, a register locking mechanism is included. These registers can be write-protected in a set of associated key registers. The MPC855T key registers are shown in Table 10-6.

Table 10-6. Key Registers

Offset	Name	Size
System Integration Timers Keys		
0x300	TBSCRK—Timebase status and control register key	32 bits
0x304	TBREFAK—Timebase reference register A key	32 bits
0x308	TBREFBK—Timebase reference register B key	32 bits
0x30C	TBK—Timebase/decrementer register key	32 bits
0x310–31F	Reserved	16 bytes
0x320	RTCSCK—Real-time clock status and control register key	32 bits
0x324	RTCK—Real-time clock register key	32 bits
0x328	RTSECK—Real-time alarm seconds key	32 bits
0x32C	RTCALK—Real-time alarm register key	32 bits
0x330–33F	Reserved	16 bytes
0x340	PISCRK—Periodic interrupt status and control register key	32 bits
0x344	PITCK—Periodic interrupt count register key	32 bits
0x348–37F	Reserved	56 bytes
Clocks and Reset Keys		
0x380	SCCRK—System clock control key	32 bits
0x384	PLPRCRK—PLL, low power and reset control register key	32 bits
0x388	RSRK—Reset status register key	32 bits
0x38C–7FF	Reserved	1140 bytes

Each register in the keep-alive power region has a key register that can be in an open or locked state. At power-on reset, all key registers are open, except for the real-time clock key registers. Each key register has an associated address in the internal memory map, as shown in Table 10-6. A write of 0x55CC_AA33 to a key register unlocks its associated SIU register; any other access (including reads or writes of any other value) to a key register locks its associated SIU register. For example, writing a 0x55CC_AA33 to the RTCK key register allows the RTC register to be written. The key registers are write-only; a read of the key register does not return the last value written.

When a register is locked, an attempt to write to it will result in a machine check exception, and will not change the value in the register. One exception to this is the timebase register (TBU and TBL), locked with TBK. A write to the timebase register when it is locked results in a software emulation exception.

Reads are allowed at all times to any of the SIU registers, regardless of whether they are locked or unlocked.

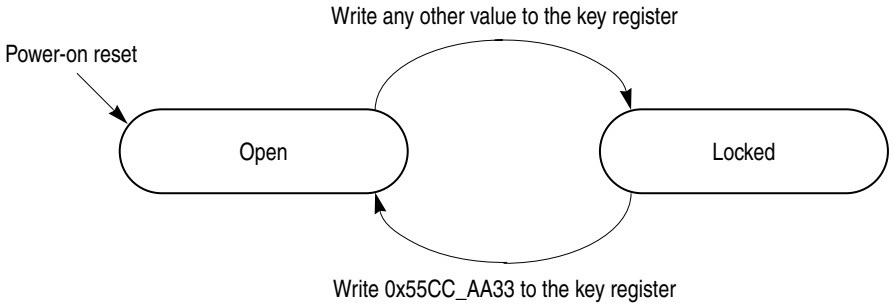


Figure 10-6. Register Lock Mechanism

For more information on key registers, see Section 14.5.7.3, “Register Lock Mechanism: Protecting SIU Registers in Power-Down Mode.”

10.5 System Configuration

The SIU module configuration register (SIUMCR) is used for configuring external bus arbitration logic, external master support, and pin multiplexing. See Section 10.4.2, “SIU Module Configuration Register (SIUMCR).”

10.5.1 Interrupt Structure

The SIU receives interrupts from internal sources, like the PIT, real-time clock, communications processor module (CPM), and the external $\overline{\text{IRQ}}$ pins. Figure 10-7 shows the MPC855T interrupt structure.

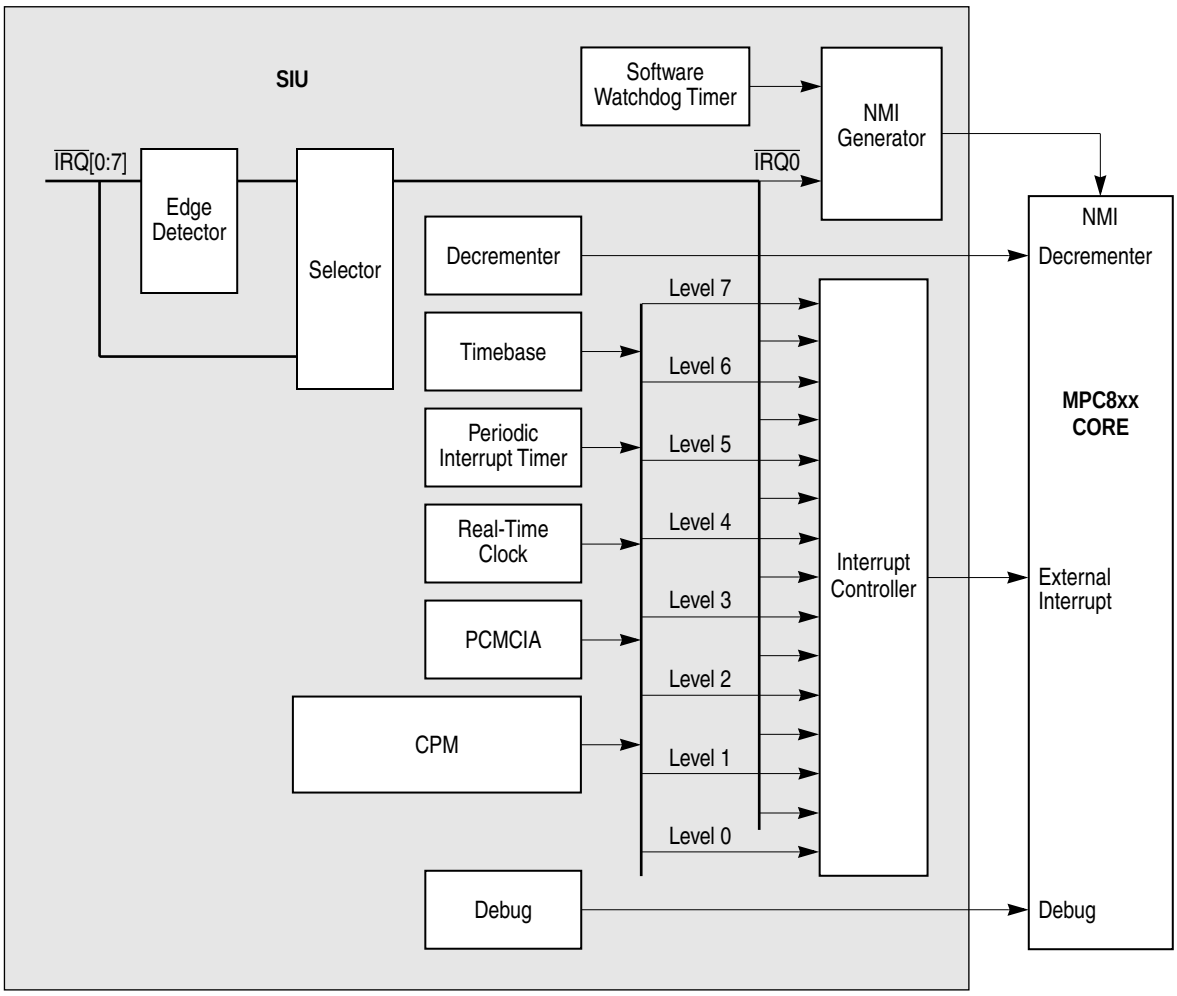


Figure 10-7. MPC855T Interrupt Structure

If programmed to generate interrupts, the software watchdog timer generates a nonmaskable system reset interrupt (NMI) to the core. Asserting the external $\overline{\text{IRQ0}}$ pin generates an NMI as well. Note that the core takes the system reset interrupt vector when an NMI is asserted and jumps to the external interrupt vector when any other interrupt is asserted by the interrupt controller. Each external $\overline{\text{IRQ}}$ pin is assigned a priority level. Each SIU internal interrupt source, generated by the CPM's interrupt controller (CPIC), can be assigned by the software to one of eight additional internal interrupt priority levels, described in Chapter 34, "CPM Interrupt Controller."

Section 10.5.3.1, "Nonmaskable Interrupts—IRQ0 and SWT," describes how $\overline{\text{IRQ0}}$ operates differently from other IRQ signals, and how the operation is configurable through SIU registers.

10.5.2 Priority of Interrupt Sources

There are eight external $\overline{\text{IRQ}}$ pins ($\overline{\text{IRQ0}}$ is essentially nonmaskable, although in a limited sense it can be masked as shown in Table 10-8) and eight interrupt levels. Asserting $\overline{\text{IRQ0}}$ causes an NMI. The other 15 interrupt sources assert a single interrupt request to the core (the external interrupt). Table 10-7 shows interrupt priorities.

Table 10-7. Priority of SIU Interrupt Sources

Number	Priority Level	Interrupt Source	Interrupt Code (SIVEC[INTC])
0	Highest	$\overline{\text{IRQ0}}$	0000_0000
1		Internal Level 0	0000_0100
2		$\overline{\text{IRQ1}}$	0000_1000
3		Internal Level 1	0000_1100
4		$\overline{\text{IRQ2}}$	0001_0000
5		Internal Level 2	0001_0100
6		$\overline{\text{IRQ3}}$	0001_1000
7		Internal Level 3	0001_1100
8		$\overline{\text{IRQ4}}$	0010_0000
9		Internal Level 4	0010_0100
10		$\overline{\text{IRQ5}}$	0010_1000
11		Internal Level 5	0010_1100
12		$\overline{\text{IRQ6}}$	0011_0000
13		Internal Level 6	0011_0100
14		$\overline{\text{IRQ7}}$	0011_1000
15	Lowest	Internal Level 7	0011_1100
16-31		Reserved	—

10.5.3 SIU Interrupt Processing

Figure 10-8 shows the general flow of SIU interrupt processing.

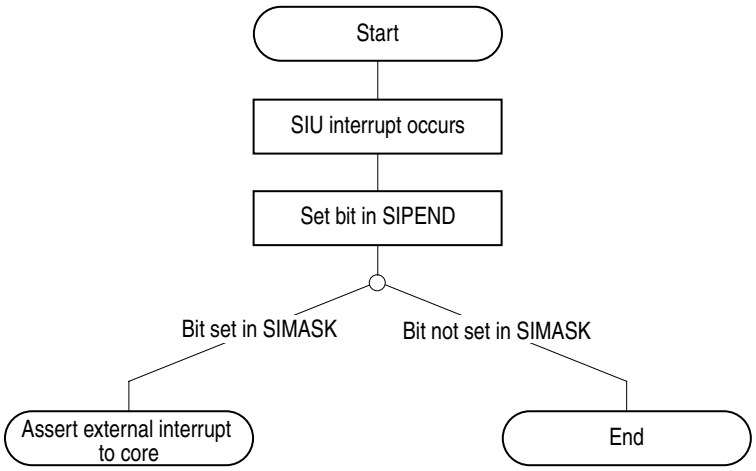


Figure 10-8. SIU Interrupt Processing

10.5.3.1 Nonmaskable Interrupts—$\overline{\text{IRQ0}}$ and SWT

Figure 10-9 is a logical representation of $\overline{\text{IRQ0}}$.

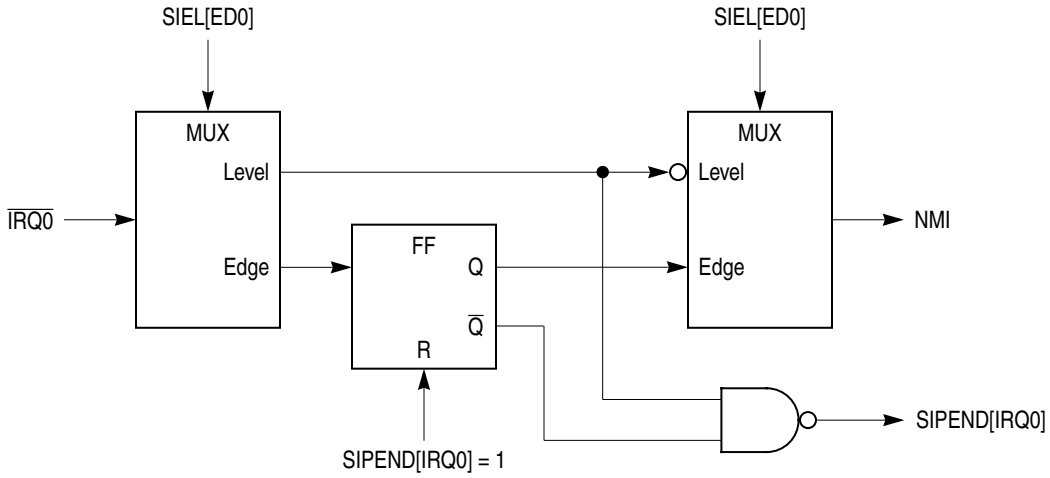


Figure 10-9. $\overline{\text{IRQ0}}$ Logical Representation

Table 10-8 describes the differences between $\overline{\text{IRQ0}}$ and other $\overline{\text{IRQ}}$ interrupts.

Table 10-8. $\overline{\text{IRQ0}}$ Versus $\overline{\text{IRQ}x}$ Operation

Functionality	$\overline{\text{IRQ0}}$	$\overline{\text{IRQ}x}$
Exception Vector	0x100	0x500
Core input	NMI	External interrupt
SIMASK	Not used, except for enabling SIVEC	Used for masking
SIVEC	Not normally used. If used, SIMASK[IRQ0] must be set.	Supplies the interrupt code so the core knows the interrupt source.

SWT (software watchdog timer) interrupts behave similarly in that they jump to the system reset vector (0x100). However, they are not affected by any interrupt controller registers.

Although NMI causes a jump to the system reset vector, no other reset action is taken. For information on recoverability of NMI, see Section 6.1.5, “Recoverability after an Exception.”

10.5.4 Programming the SIU Interrupt Controller

The SIU’s interrupt controller includes the SIU interrupt pending register (SIPEND), SIU interrupt mask register (SIMASK), SIU interrupt edge/level register (SIEL), and SIU interrupt vector register (SIVEC) registers. These are described in the following sections.

10.5.4.1 SIU Interrupt Pending Register (SIPEND)

SIU interrupt pending register (SIPEND) bits, shown in Figure 10-10, correspond to interrupt requests. This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

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	LVL7
Reset	0000_0000_0000_0000															
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	—															
Reset	xxxx_xxxx_xxxx_xxxx															
R/W	R/W															
Addr	(IMMR & 0xFFFF0000) + 0x012															

Figure 10-10. SIU Interrupt Pending Register (SIPEND)

Table 10-9 describes SIPEND fields.

Table 10-9. SIPEND Field Descriptions

Bits	Name	Description
0, 2, 4, 6, 8, 10, 12, 14	IRQ*n*	Interrupt request 0–7. Indicate whether an edge-triggered interrupt is pending. 0 The appropriate interrupt is not pending. 1 The appropriate interrupt is pending.
1, 3, 5, 7, 9, 11, 13, 15	LVL*n*	Level 0–7. When set, these bits indicate a pending level interrupt of corresponding value. 0 The appropriate interrupt is not pending. 1 The appropriate interrupt is pending.
16–31	—	Reserved, should be cleared.

The LVL[0–7] bits are associated with internal exceptions, and when set indicate that an interrupt service is requested if they are not masked by the corresponding SIMASK bit. These bits reflect the status of the internal requesting device and are cleared when the appropriate actions are software-initiated in the device. Writing to LVL*n* bits has no effect.

The IRQ[0–7] bits are associated with the $\overline{\text{IRQ}}$[0–7] signals, and their function depends on the sensitivity defined for them in SIEL; see Section 10.5.4.3, “SIU Interrupt Edge/Level Register (SIEL).”

- When an $\overline{\text{IRQ}}$ pin is defined as a level interrupt (SIEL[ED*n*] = 0), the corresponding IRQ bit behaves like an LVL bit.
- If an $\overline{\text{IRQ}}$ pin is defined as an edge interrupt (SIEL[ED*n*] = 1), the corresponding bit being set indicates that a falling edge was detected on the line and are reset by writing ones to them.

Note that $\overline{\text{IRQ0}}$ can be masked in only a very limited sense. If SIEL[ED0] = 1, edge-sensitive, and SIPEND[IRQ0] is not cleared in the interrupt service routine, further assertions of $\overline{\text{IRQ0}}$ are masked.

10.5.4.2 SIU Interrupt Mask Register (SIMASK)

Bits in SIMASK correspond to the interrupt request bits in SIPEND. Setting SIMASK bits enable the generation of interrupt requests to the core. SIMASK is updated by the software, which must determine which interrupt sources are enabled at a given time. This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

.

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	0000_0000_0000_0000															
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	—															
Reset	xxxx_xxxx_xxxx_xxxx															
R/W	R/W															
Addr	(IMMR & 0xFFFF0000) + 0x016															

Figure 10-11. SIU Interrupt Mask Register (SIMASK)

Table 10-10 describes SIMASK fields.

Table 10-10. SIMASK Field Descriptions

Bits	Name	Description
0	IRM0	Interrupt request mask 0. Enables/disables updating SIVEC[INTC]. $\overline{\text{IRQ0}}$ generates an NMI regardless of this bit.
1, 3, 5, 7, 9, 11, 13, 15	LVM*n*	Level mask 0–7. When set, these bits enable an internal interrupt request to be generated. 0 Disable generation of an interrupt request bit in SIPEND. 1 Enable generation of an interrupt request bit in SIPEND.
2, 4, 6, 8, 10, 12, 14	IRM*n*	Interrupt request mask 1–7. When set, these bits enable an $\overline{\text{IRQ}}$ interrupt request to be generated. 0 Disable generation of an interrupt request bit in SIPEND. 1 Enable generation of an interrupt request bit in SIPEND.
16–31	—	Reserved, should be cleared.

The following procedure prevents possible interrupt errors when modifying mask registers, such as SIMASK:

1. Clear MSR[EE]. (Disable external interrupts to the core.)
2. Modify the mask register.
3. Set MSR[EE]. (Enable external interrupts to the core.)

This mask modification procedure ensures that an already pending interrupt is not masked before being serviced.

10.5.4.3 SIU Interrupt Edge/Level Register (SIEL)

Bits in SIEL, shown in Figure 10-12, define interrupts as edge- or level-triggered and enable/disable their use as wake-up signals in low-power mode. This register is affected by $\overline{\text{HRESET}}$ but is not affected by $\overline{\text{SRESET}}$.

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	0000_0000_0000_0000															
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	—															
Reset	xxxx_xxxx_xxxx_xxxx															
R/W	R/W															
Addr	(IMMR & 0xFFFF0000) + 0x01A															

Figure 10-12. SIU Interrupt Edge/Level Register (SIEL)

Table 10-11 describes SIEL fields.

Table 10-11. SIEL Field Descriptions

Bits	Name	Description
0, 2, 4, 6, 8, 10, 12, 14	ED*n*	Edge detect 0–7. 0 A low logical level in the $\overline{\text{IRQ}}$ signal indicates an interrupt request. 1 A falling edge in the corresponding $\overline{\text{IRQ}}$ signal indicates interrupt request.
1, 3, 5, 7, 9, 11, 13, 15	WM*n*	Wake-up mask 0–7 0 Not allowed to exit from low-power mode. 1 Low-level detection in $\overline{\text{IRQ}}$*n* allows the MPC855T to exit or wake up from low-power mode.
16–31	—	Reserved, should be cleared.

10.5.4.4 SIU Interrupt Vector Register (SIVEC)

The SIU interrupt vector register (SIVEC) is shown in Figure 10-13. This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	INTC								—							
Reset	xx11_11xx_xxxx_xxxx															
R/W	R															
Addr	(IMMR & 0xFFFF0000) + 0x01C															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—															
Reset	xxxx_xxxx_xxxx_xxxx															
R/W	R															
Addr	(IMMR & 0xFFFF0000) + 0x01E															

Figure 10-13. SIU Interrupt Vector Register (SIVEC)

Table 10-12 describes SIVEC fields.

Table 10-12. SIVEC Field Descriptions

Bits	Name	Description
0–7	INTC	Interrupt code. Indicates the highest priority pending interrupt; equals the interrupt number times 4, as shown in Table 10-7.
8–31	—	Reserved, should be cleared.

SIVEC[INTC] represents the unmasked interrupt source of the highest priority level. When SIVEC is read as a byte, a branch table can be used in which each entry contains one instruction (branch). The interrupt code is the interrupt number times 4, which allows indexing into the table. When read as a half word, each entry can contain a full routine of up to 256 instructions; see Figure 10-14 and Table 10-7.

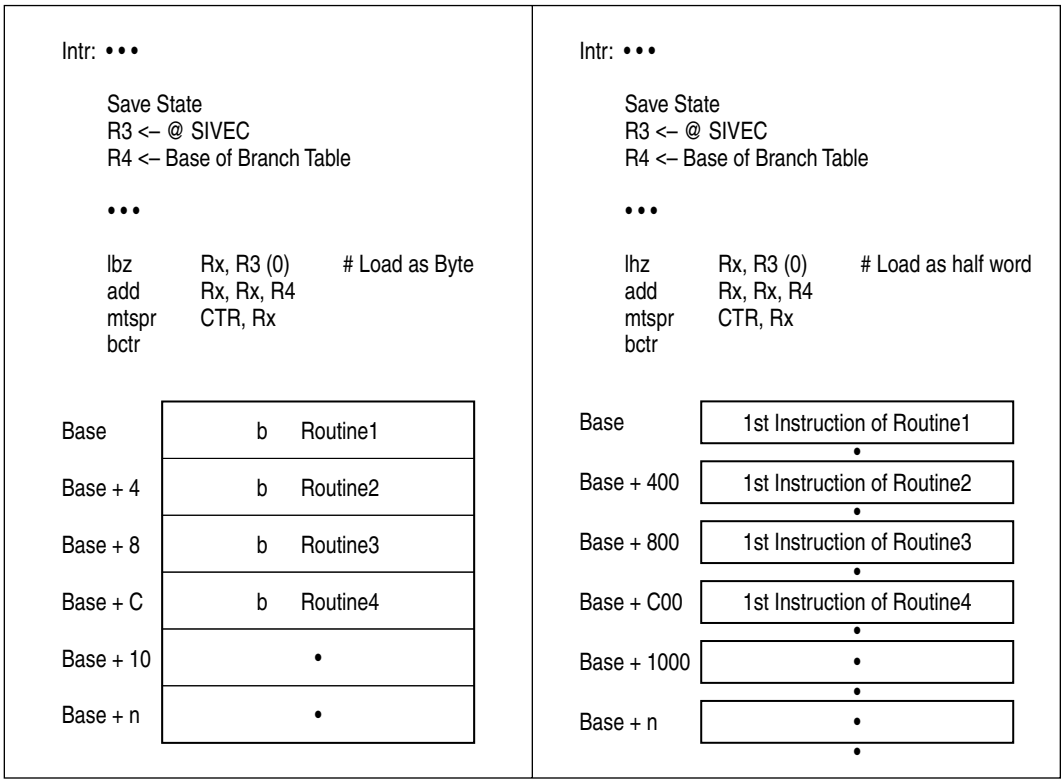


Figure 10-14. Interrupt Table Handling Example

The interrupt to be serviced can be determined by reading SIVEC[INTC]. For example, if $\overline{\text{IRQ3}}$, level 3, and $\overline{\text{IRQ6}}$ interrupts occur simultaneously and $\overline{\text{IRQ3}}$ is masked, INTC = 0b0001_1100 (0x1C), indicating that the level 3 interrupt should be handled.

Note that SIVEC[INTC] contains the encoding for a level-7 interrupt (see Table 10-7) by default, even when no interrupts are pending. Thus, polling SIVEC when all interrupts are masked returns the level-7 vector. Therefore, the level-7 interrupt vector may indicate a spurious interrupt in the following cases:

- Polling SIVEC returns a level 7 interrupt, but nothing is programmed to interrupt at level 7.
- Polling SIVEC returns a level 7 interrupt, but SIPEND[LV7] is not set (assuming something is programmed to interrupt at level 7).

10.6 The Bus Monitor

Control of the bus monitor is provided in the SYPCR. The bus monitor ensures that each bus cycle initiated by the MPC855T terminates within a reasonable time. The MPC855T's bus monitor does not monitor accesses initiated by external masters. At the start of the transfer start signal ($\overline{\text{TS}}$), the monitor begins counting and stops when transfer acknowledge ($\overline{\text{TA}}$), retry ($\overline{\text{RETRY}}$) or transfer error ($\overline{\text{TEA}}$) is asserted. For burst cycles, this action is also

performed between subsequent $\overline{TA}$ assertions for each data beat. If the monitor times out, the bus monitor terminates the cycle by internally asserting $\overline{TEA}$. 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 is 2,040 system clocks. The bus monitor is always active when FRZ is asserted or when a debug mode request is pending, regardless of the state of the SYPCR[BME] bit.

Note that if the bus monitor is disabled, transfer errors do not cause $\overline{TEA}$ to be asserted.

10.7 Software Watchdog Timer

The SIU provides the software watchdog timer (SWT) option that prevents system lockup when software gets trapped in loops without a controlled exit. The software watchdog timer is enabled after $\overline{HRESET}$ to automatically generate a $\overline{HRESET}$ if it times out. If the software watchdog timer is unneeded, clear SYPCR[SWE] to disable it. If it is used, the software watchdog timer requires a special service sequence to be executed periodically; otherwise, the watchdog timer times out and issues a reset or an NMI, which is programmed by SYPCR[SWRI]. Once SYPCR is written by the software, SYPCR[SWE] cannot be changed. See Section 10.4.3, “System Protection Control Register (SYPCR).” To service the software watchdog timer, follow these steps:

1. Write 0x556C to the software service register. (SWSR)
2. Write 0xAA39 to the SWSR.

This sequence clears the watchdog timer and the timing process repeats. If a value other than 0x556C or 0xAA39 is used, 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. See Figure 10-15.

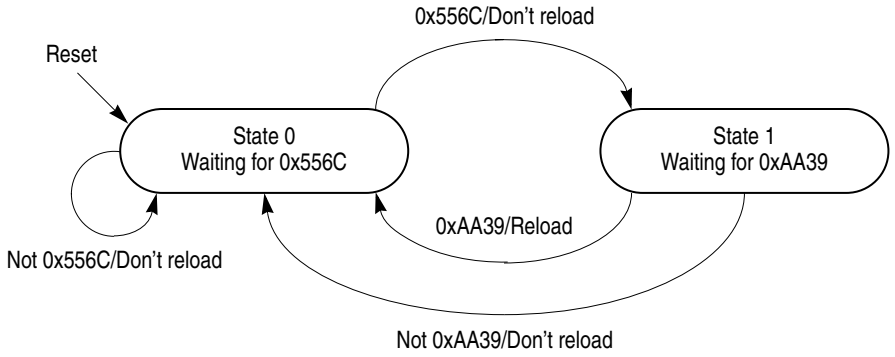


Figure 10-15. Software Watchdog Timer Service State Diagram

The decrementer begins counting when it is loaded with a value from the SWTC field. This value is then loaded into a 16-bit down-counter clocked by the system clock. When necessary, an additional divide by 2,048 prescaler is used. After the timer reaches 0x0, a software watchdog expiration request is issued to the reset or NMI control logic. At reset,

the value in SWTC is set to the maximum value and is loaded into the software watchdog down-counter, starting the process.

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 provides a selectable range for the timeout period. Figure 10-16 shows the method for handling this need. When a new value is loaded into SWTC, the software watchdog timer is not updated until the servicing sequence is written to SWSR. If the SWE bit is loaded with a zero, the modulus counter will not count.

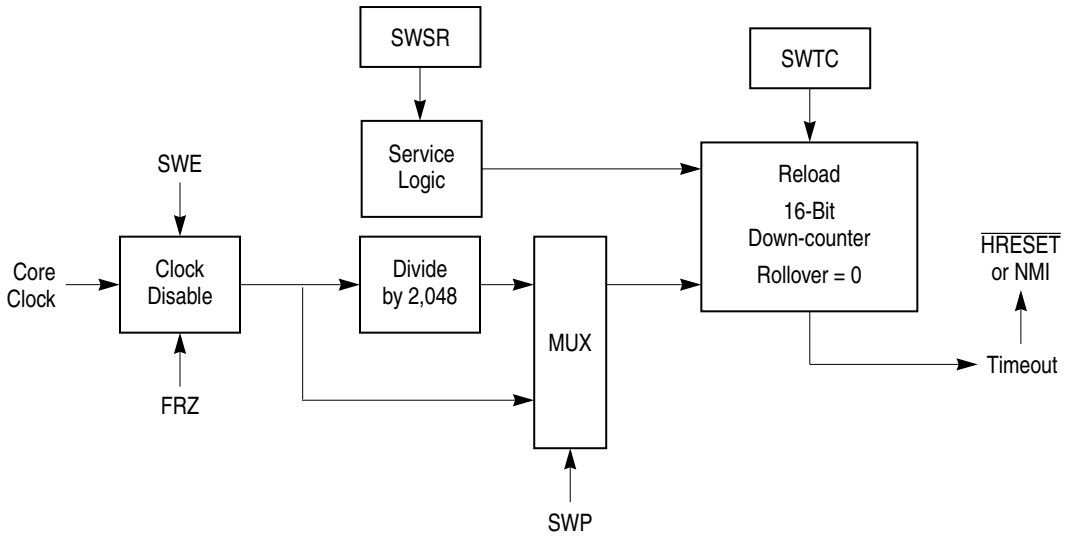


Figure 10-16. Software Watchdog Timer Block Diagram

10.7.1 Software Service Register (SWSR)

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. This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	SEQ															
Reset	0000_0000_0000_0000															
R/W	W															
Addr	(IMMR & 0xFFFF0000) + 0x00E															

Figure 10-17. Software Service Register (SWSR)

Table 10-13 describes SWSR fields.

Table 10-13. SWSR Field Descriptions

Bits	Name	Description
0–15	SEQ	Sequence. This field is the pattern that is used to control the state of the software watchdog timer.

10.8 The Decrementer

A PowerPC-defined 32-bit decrementing counter supports the decrementer interrupt. In the MPC855T, the decrementer is clocked by TMBCLK, so TBSCR[TBE] must be set for the decrementer to start. The timebase and decrementer counters are driven by TMBCLK:

$$T_{dec} = \frac{2^{32}}{(F_{tmbclk})}$$

The state of the decrementer is not affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$, so it should be initialized by software. Note, however, that it is disabled and reset by $\overline{\text{PORESET}}$. 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 is unaffected when read.
- When DEC[0] changes from 0 to 1, an interrupt request is signaled. If a previous decrementer interrupt request was made, only one interrupt is reported.
- Explicitly changing DEC[0] from 0 to 1 in software signals an interrupt request.

A decrementer interrupt is also sent to the power-down wake-up logic, so the core can waken from power-down mode. A decrementer exception causes a pending interrupt request in the core, which is cleared automatically when the decrementer interrupt is taken, Table 10-14 shows some decrementer periods available, assuming a 4-MHz oscillator.

Table 10-14. Decrementer Timeout Values

Count Value	Timeout
0	1 µs
9	10. µs
99	100. µs
999	1.0 ms
9999	10.0 ms

Count Value	Timeout
999999	1.0 s
9999999	10.0 s
99999999	100.0 s
999999999	1,000 s
FFFFFFFF (hex)	4,295 s

10.8.1 Decrementer Register (DEC)

The decrementer register (DEC) is an SPR as defined in the PowerPC architecture. It can be read or written to by **mfspr** or **mtspr**. DEC is powered by KAPWR and continues

counting when KAPWR is applied. Control of the decrementer is provided in the TBSCR. The decrementer and timebase use TMBCLK. Note that DEC is a keyed register. It must be unlocked in TBK before it can be written.

Bit	0	1	2	3	4	5	6	7	8	9	…	30	31
Field	DEC												
Reset	—												
R/W	R/W												
SPR	22												

Figure 10-18. Decrementer Register (DEC)

Table 10-15 describes the DEC register.

Table 10-15. DEC Field Descriptions

Bits	Name	Description
0–31	DEC	Decrementer. These bits are used by a down counter to cause decrementer interrupts. Reading DEC always returns the current count value from the down counter.

10.9 The Timebase

The timebase is a 64-bit free-running binary counter as defined in the PowerPC architecture. For the MPC855T, the timebase is clocked by TMBCLK. The timebase period is as follows:

$$T_{TB} = \frac{2^{64}}{F_{tmbclk}}$$

The timebase is unaffected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ and should be initialized by software. Note, however, that it is disabled and reset by $\overline{\text{PORESET}}$. The entire timebase cannot be accessed with a single instruction; **mttb** and **mftb** access the lower half of the timebase and **mttbu** and **mftbu** access the upper half. A maskable interrupt is generated when the timebase count reaches a value programmed in one of the reference registers, TBREFA and TBREFB; two status bits indicate which one caused the interrupt.

10.9.1 Timebase Register (TBU and TBL)

The timebase register (TB) holds a 64-bit integer that is incremented periodically. It is implemented in two parts, time base upper and time base lower (TBU and TBL). There is no automatic initialization of TB, therefore, system software must perform this initialization. The contents of TB can be written by **mtspr** and read by **mftb** or **mftbu** instruction. Figure 10-19 shows TBU. Note that the TBU and TBL are keyed registers. They must be unlocked in TBK before they can be written.

Bit	0	1	2	3	4	5	6	7	8	9	…	30	31
Field	TBU												
Reset	—												
R/W	R/W												
SPR	269 (Read)/285 (Write)												

Figure 10-19. Timebase Upper Register (TBU)

Table 10-16 describes TBU fields.

Table 10-16. TBU Field Descriptions

Bits	Name	Description
0–31	TBU	Timebase upper. The value in this field is used as an upper part of the timebase counter.

Figure 10-20 shows TBL.

Bit	0	1	2	3	4	5	6	7	8	9	…	30	31
Field	TBL												
Reset	—												
R/W	R/W												
SPR	268 (Read)/284 (Write)												

Figure 10-20. Timebase Lower Register (TBL)

Table 10-17 describes TBL fields.

Table 10-17. TBL Field Descriptions

Bits	Name	Description
0–31	TBL	Timebase lower. The value in this field is used as the lower part of the timebase register.

10.9.2 Timebase Reference Registers (TBREFA and TBREFB)

TBREFA and TBREFB are associated with TBL. When the contents of TBL matches a reference register, a reference event is signaled in TBSCR[REFA] or TBSCR[REFB]. These events can generate interrupts, if enabled. Note that TBREFA and TBREFB are keyed registers. They must be unlocked in TBREFAK and TBREFBK before they can be written.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	TBREFA/TBREFB															
Reset	—															
R/W	R/W															
Addr	TBREFA (IMMR & 0xFFFF0000) + 0x204/TBREFB (IMMR & 0xFFFF0000) + 0x208															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	TBREFA/TBREFB															
Reset	—															
R/W	R/W															
Addr	TBREFA (IMMR & 0xFFFF0000) + 0x206/TBREFB (IMMR & 0xFFFF0000) + 0x20A															

Figure 10-21. Timebase Reference Registers (TBREFA and TBREFB)

These registers are affected by $\overline{\text{HRESET}}$ but are not affected by $\overline{\text{SRESET}}$. Table 10-18 describes TBREFA/TBREFB fields.

Table 10-18. TBREFA/TBREFB Field Descriptions

Bits	Name	Description
0–31	TBREFA	Timebase reference A. Represents the 32-bit reference value for TBL.
0–31	TBREFB	Timebase reference B. Represents the 32-bit reference value for TBL

10.9.3 Timebase Status and Control Register (TBSCR)

The timebase status and control register (TBSCR) controls the timebase count enable and interrupt generation. It is also is used for reporting the interrupt sources, and it can be read at any time. Status bits are cleared by writing ones; writing zeros has no effect. Note that TBSCR is a keyed register. It must be unlocked in TBSCRK before it can be written.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	TBIRQ								REFA	REFB	—		REFAE	REFBE	TBF	TBE
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	(IMMR & 0xFFFF0000) + 0x200															

Figure 10-22. Timebase Status and Control Register (TBSCR)

This register is affected by HRESET but is not affected by SRESET. Table 10-19 describes TBSCR fields.

Table 10-19. TBSCR Field Descriptions

Bits	Name	Description
0–7	TBIRQ	Timebase interrupt request. Determines interrupt priority level of the timebase. To specify a certain level, the appropriate bit should be set.
8	REFA	Reference interrupt status. If set, indicates that a match was detected between the corresponding reference register (TBREFA for REFA and TBREFB for REFB) and the TBL. REFA and REFB are cleared by writing ones.
9	REFB	
10–11	—	Reserved, should be cleared.
12	REFAE	Reference interrupt enable. If asserted, the timebase generates an interrupt on assertion of REFA or REFB. Otherwise, the interrupt is disabled.
13	REFBE	
14	TBF	Timebase freeze enable 0 The timebase and decrementer are unaffected. 1 The FRZ signal stops the timebase and decrementer.
15	TBE	Timebase enable 0 Disables timebase and decrementer operation. 1 Enables timebase and decrementer operation.

10.10 The Real-Time Clock

The real-time clock is a 45-bit counter, clocked by PITRTCLK, to provide time-of-day to the operating system and application software. The counter is not affected by HRESET, SRESET, or PORESET and operates in all low-power modes. It must be initialized by 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 each second. A control and status register is used to selectively enable or disable functions and report the interrupt source. The real-time clock registers (RTCSC, RTC, RTSEC, and RTCAL) can be protected (locked) from accidental writes after PORESET through the use of key registers (RTCSCK, RTCK, RTSECK, and RTCALK), which are described in Section 10.4.5, “Register Lock Mechanism.” To unlock a register, write the key word 0x55CC_AA33 to the key registers.

Note that the real-time clock will count in seconds only if PITRTCLK is supplied by a 32.768 KHz or 38.4 KHz source.

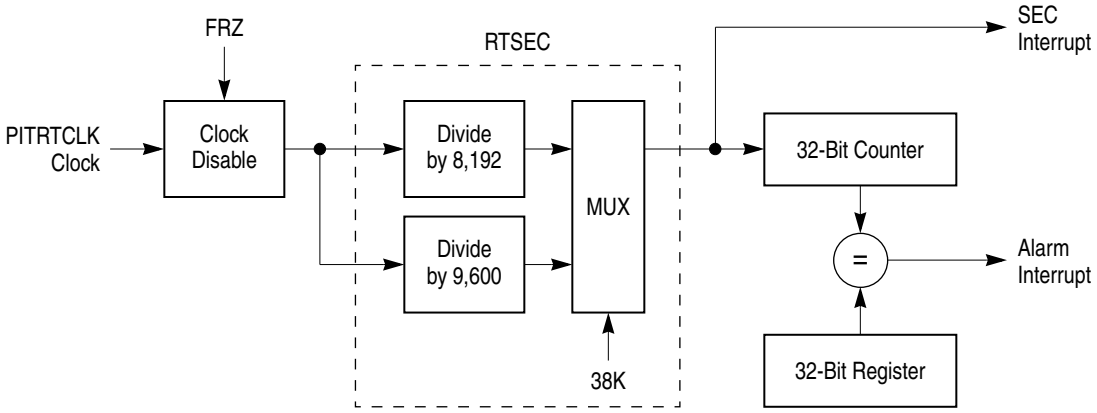


Figure 10-23. Real-Time Clock Block Diagram

10.10.1 Real-Time Clock Status and Control Register (RTCSC)

The real-time clock status and control register (RTCSC) is used to enable the different real-time clock functions and for reporting interrupt sources. Status bits are cleared by writing ones; writing zeros has no effect. Note that RTCSC is a keyed register; it must be unlocked in RTCSCK before it can be written.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	RTCIRQ								SEC	ALR	—	38K	SIE	ALE	RTF	RTE
Reset	0000_0000								0	0	0	—	0	0	0	—
R/W	R/W															
Addr	(IMMR & 0xFFFF0000) + 0x220															

Figure 10-24. Real-Time Clock Status and Control Register (RTCSC)

This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Table 10-20 describes RTCSC fields.

Table 10-20. RTCSC Field Descriptions

Bits	Name	Description
0–7	RTCIRQ	Real-time clock interrupt request. These bits control the real-time clock's internal interrupt priority level.
8	SEC	Once-per-second interrupt. Set every second; should be cleared by software.
9	ALR	Alarm interrupt. Status bit set when the value of the real-time clock equals the value in RTCAL.
10	—	Reserved, should be cleared.
11	38K	Real-time clock source select. Software must set 38K for the proper timing of a second. 0 Assumes that PITRTCLK is driven by a 32.768-KHz crystal 1 Assumes that PITRTCLK is driven by a 38.4-KHz crystal.
12	SIE	Seconds interrupt enable. If set, the real-time clock generates an interrupt when SEC is set.

Table 10-20. RTCSC Field Descriptions (continued)

Bits	Name	Description
13	ALE	Alarm interrupt enable. If set, the real-time clock generates an interrupt when ALR is set.
14	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.
15	RTE	Real-time clock enable. If set, real-time clock timers are enabled.

10.10.2 Real-Time Clock Register (RTC)

The 32-bit real-time clock register (RTC) contains the current value of the real-time clock. The maximum value is approximately 136 years. Note that RTC is a keyed register. It must be unlocked in RTCK before it can be written.

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															

Figure 10-25. Real-Time Clock Register (RTC)

This register is not affected by $\overline{\text{HRESET}}$ but is affected by $\overline{\text{SRESET}}$. Table 10-21 describes the RTC.

Table 10-21. RTC Field Description

Bits	Name	Description
0–31	RTC	Real-time clock. Represents time measured in seconds. Each unit represents one second.

10.10.3 Real-Time Clock Alarm Register (RTCAL)

The real-time clock alarm register (RTCAL) is an alarm reference register. When RTC increments to the value stored in this register, an alarm interrupt is generated. Note that RTCAL is a keyed register. It must be unlocked in RTCALK before it can be written.

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															

Figure 10-26. Real-Time Clock Alarm Register (RTCAL)

This register is not affected by $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$. Table 10-22 describes RTCAL fields.

Table 10-22. RTCAL Field Descriptions

Bits	Name	Description
0–31	ALARM	Alarm reference counter. Indicates that an alarm interrupt will be generated this field matches corresponding RTC bits. The alarm has a 1-second resolution.

10.10.4Real-Time Clock Alarm Seconds Register (RTSEC)

RTSEC, shown in Figure 10-27, is a down-counter that decrements once per PITRTCLK tick. Note that RTSEC is a keyed register. It must be unlocked in RTSECK before it can be written.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	COUNTER														—	
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	—															
Reset	—															
R/W	R/W															
Addr	(IMMR & 0xFFFF0000) + 0x22A															

Figure 10-27. Real-Time Clock Alarm Seconds Register (RTSEC)

This register is not affected by $\overline{\text{HRESET}}$ but is affected by $\overline{\text{SRESET}}$. Table 10-23 describes RTSEC fields.

Table 10-23. RTSEC Field Descriptions

Bits	Name	Description
0–13	COUNTER	Counter bits (fraction of a second). Bit 13 is always the lsb of the count. It either resets at 8192 or at 9600, as programmed.
14–31	—	Reserved; should be cleared.

Under normal conditions (RTCSC[38K] = 0), PITRTCLK is assumed to be 8192 Hz (4.192 MHz/512 or 32.768 KHz/4). When RTSEC counts down to zero, RTC is incremented. Thus, RTC contains the time in seconds and RTSEC functions as a divider. For a 38.4-KHz crystal (instead of 32.768 KHz), RTCSC[38K] should be set to make RTSEC reset every 9600 ticks instead of 8192.

10.11 Periodic Interrupt Timer (PIT)

The PIT, shown in Figure 10-28, consists of a 16-bit counter clocked by a PITRTCLK clock supplied by the clock module. The PIT is not affected by $\overline{\text{HRESET}}$ and $\overline{\text{RESET}}$; however, it is disabled and reset by $\overline{\text{PORESET}}$. It decrements to zero when loaded with a value from the PIT count register (PITC) and after the timer reaches zero, PS is set and an interrupt is generated if PIE is a 1. At the next input clock edge, the PITC value is loaded into the counter and the process repeats. When a new value is loaded into PITC, the PIT is updated, the divider is reset, and the counter starts counting. If the PS bit is set, an interrupt is generated at the interrupt controller that remains pending until PS is cleared. If PS is set again, before being cleared, the interrupt remains pending until PS is cleared. Any write to PITC stops the current countdown and the count resumes with a new value in the PITC. If the PTE bit is not set, the PIT is unable to count and retains the old count value. Reading the PIT does not affect it.

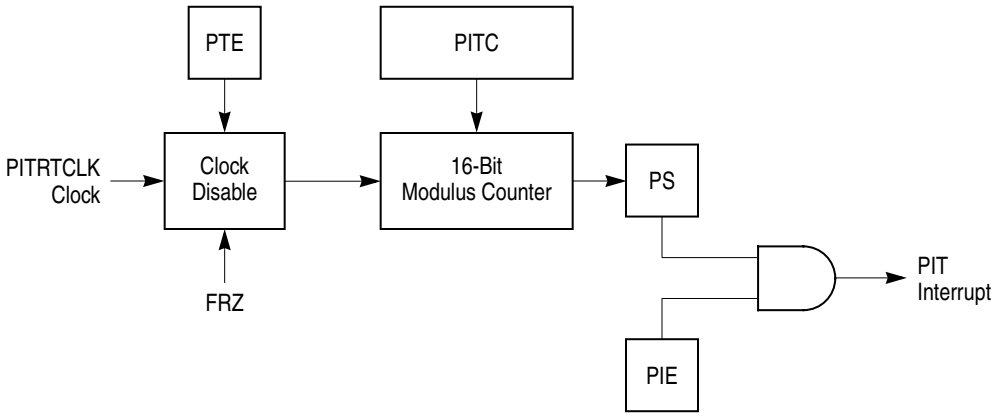


Figure 10-28. Periodic Interrupt Timer Block Diagram

The time-out period is calculated as follows:

Solving this equation using a 32.768-KHz external clock gives:

$$PIT_{period} = \frac{PITC + 1}{F_{pitrtclk}} = \frac{PITC + 1}{\left(\frac{ExternalClock}{1 \text{ or } 128}\right) \div 4}$$

$$PITperiod = \frac{PITC + 1}{8192}$$

This gives a range from 122 µs (PITC = 0x0000) to 8 seconds (PITC = 0xFFFF).

10.11.1 Periodic Interrupt Status and Control Register (PISCR)

The periodic interrupt status and control register (PISCR), shown in Figure 10-29, contains the interrupt request level and status bits. It also controls the 16 bits to be loaded in a modulus counter. Note that PISCR is a keyed register. It must be unlocked in PISCRK before it can be written.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	PIRQ								PS	—				PIE	PITF	PTE
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	(IMMR & 0xFFFF0000) + 0x240															

Figure 10-29. Periodic Interrupt Status and Control Register (PISCR)

This register is affected by $\overline{\text{HRESET}}$ but is not affected by $\overline{\text{SRESET}}$. Table 10-24 describes PISCR fields.

Table 10-24. PISCR Field Descriptions

Bits	Name	Description
0–7	PIRQ	Periodic interrupt request level. Configures internal interrupt levels for periodic interrupts. Figure 10-7 shows interrupt request levels.
8	PS	Periodic interrupt status. Can be cleared by writing a 1 to it (zero has no effect). 0 The PIT is unaffected. 1 The PIT has issued an interrupt.
9–12	—	Reserved, should be cleared.
13	PIE	Periodic interrupt enable 0 Disables the PS bit. 1 Enables the PS bit to generate an interrupt.

Table 10-24. PISCR Field Descriptions (continued)

Bits	Name	Description
14	PITF	PIT freeze enable 0 The PIT is unaffected by the FRZ signal. 1 The FRZ signal stops the PIT.
15	PTE	Periodic timer enable 0 The PIT is disabled. 1 The PIT is enabled.

10.11.2PIT Count Register (PITC)

PITC, shown in Figure 10-30, contains a 16-bit value to be loaded into the periodic interrupt down counter. Note that PITC is a keyed register. It must be unlocked in PITCK before it can be written.

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	—															
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	(IMMR & 0xFFFF0000) + 0x246															

Figure 10-30. PIT Count Register (PITC)

This register is not affected by $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$. Table 10-25 describes PITC fields.

Table 10-25. PITC Field Descriptions

Bits	Name	Description
0–15	PITC	PIT count. Contains the count for the periodic timer. Setting this field to 0xFFFF selects the maximum count period.
16–31	—	Reserved, should be cleared.

10.11.3PIT Register (PITR)

The PIT register (PITR) is a read-only register that shows the current value in the periodic interrupt down counter. Writes to PITR do not affect it; reads do not affect the counter.

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	—															
Reset	-															
R/W	R															
Addr	(IMMR & 0xFFFF0000) + 0x24A															

Figure 10-31. PIT Register (PITR)

Table 10-26 describes PITR fields.

Table 10-26. PITR Field Descriptions

Bits	Name	Description
0–15	PIT	Periodic interrupt timing count. Holds the current count remaining for the periodic timer. Writes do not affect PIT.
16–31	—	Reserved, should be cleared.

10.12 General SIU Timers Operation

The following sections provide detailed information on the operation of the SIU timers.

10.12.1 Freeze Operation

The external FRZ signal is asserted as a result of entry into debug mode, or as a result of actions performed by a software monitor debugger as described in Section 44.4.1, “Freeze Indication.” When the FRZ signal is asserted, the clocks to the software watchdog, PIT, 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.

10.12.2 Low-Power Stop Operation

When the MPC8xx 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 PIT, decrementer, and timebase are not affected by low-power modes and continue to run at their respective frequencies. These timers can generate an interrupt to bring the MPC855T out of the low-power modes.

Chapter 11
Reset

The reset block has 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 signals are initialized only on hard reset. Soft reset initializes the internal logic while maintaining the system configuration. The system configuration includes the SIU pin configuration, the parallel I/O configuration, and the memory controller configuration. Table 11-1 shows the reset responses of the MPC855T.

Table 11-1. MPC855T Reset Responses

Reset Source	Reset Effect						
	Reset Logic and PLL States Reset	System Configuration[1] Reset	Clock Module Reset	$\overline{\text{HRESET}}$ Driven	Debug Port Config	Other Internal Logic[2] Config Reset	$\overline{\text{SRESET}}$ Driven
Power-on reset	Yes	Yes	Yes	Yes	Yes	Yes	Yes
External hard reset, loss-of-lock, software watchdog, checkstop, debug port hard reset	No						
JTAG reset, external soft reset, debug port soft reset		No	No	No			

[1]Includes SIU pin configuration, the parallel I/O configuration and the memory controller configuration
[2]Includes all other CPM and core logic not explicitly noted elsewhere in the table

11.1 Types of Reset

The MPC855T has several sources of input 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

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.

11.1.1 Power-On Reset

Power-on reset of the MPC855T is accomplished through the $\overline{\text{PORESET}}$ input signal. The $\overline{\text{PORESET}}$ signal must be externally asserted following initial power-up, or when the keep-alive power (KAPWR) voltage falls below the minimum required for proper system operation in systems providing a power-down mode. When $\overline{\text{PORESET}}$ is asserted the MODCK bits are sampled to configure SCCR[RTDIV] and SCCR[RTSEL]. The phase-locked loop multiplication factor is configured for default operation in the PLPRCR register. When $\overline{\text{PORESET}}$ is negated, the MODCK values are sampled and internally latched. To ensure proper operation, $\overline{\text{PORESET}}$ should be asserted for a minimum of 3 µs. After sampling the assertion of $\overline{\text{PORESET}}$, the MPC855T enters the power-on reset state and stays there until both of the following events occur:

- The internal PLL enters the lock state and the system clock is active.
- $\overline{\text{PORESET}}$ 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 both $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ for 512 clock cycles. After 512 cycles elapse, the MPC855T's configuration is sampled from the data signals and the core stops internally asserting both $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. To ensure prompt negation, external pull-up resistors should be provided to drive $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ high. After $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ are internally negated, a 16-cycle period passes before the presence of an external (hard/soft) reset will be sampled. See Section 11.3.1, "Hard Reset," for more information.

11.1.2 External Hard Reset

The hard reset ($\overline{\text{HRESET}}$) signal is a bidirectional, active low, open-collector I/O signal. The MPC855T can only sample an external assertion of $\overline{\text{HRESET}}$ if it occurs while the MPC855T is not internally asserting $\overline{\text{HRESET}}$. While $\overline{\text{HRESET}}$ is asserted, $\overline{\text{SRESET}}$ is also asserted.

11.1.3 Internal Hard Reset

When the core initiates a hard reset it asserts the $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ signals for 512 cycles. After 512 clock cycles the data signals are sampled, initial configuration is established, and the core stops driving the $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ signals. Following the negation of $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ a 16-cycle period passes before an external hard or soft reset will be sampled. Note that external pull-up resistors should be provided to drive $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ high. See Section 11.3.1, “Hard Reset,” for more information.

The causes of internal hard reset are as follows:

- PLL loss of lock
- Software watchdog reset
- Checkstop reset
- Debug port hard reset

The following sections describe the events that can initiate an internal assertion of $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

11.1.3.1 PLL Loss of Lock

If the PLL experiences a loss of lock erroneous external bus operation may occur. Erroneous operation can also occur if devices with a PLL use the core CLKOUT signal as a driver. If PLPRCR[LOLRE] = 1 and a PLL loss-of-lock event occurs, an internal hard reset sequence is generated. See Section 14.6.2, “PLL, Low-Power, and Reset Control Register (PLPRCR).”

NOTE

The PLL loss of lock detection does not have a specification for the detection threshold. Therefore it should used solely as a debug tool and not in production systems. Characterization of the threshold value over temperature and operating voltages has shown that the threshold can be triggered when clock out to clock in phase differences is 1.8 ns or more.

11.1.3.2 Software Watchdog Reset

When the core watchdog counter decrements to zero, a software watchdog reset is asserted generating an internal hard reset sequence. Note that this is the default response; that is, an NMI to the core can be issued instead of a hard reset, and the timer can be disabled. See Section 10.7, “Software Watchdog Timer.”

11.1.3.3 Checkstop Reset

If the core enters a checkstop state and PLPRCR[CSR] = 1, the checkstop reset is asserted generating an internal hard reset sequence. See Section 14.6.2, “PLL, Low-Power, and Reset Control Register (PLPRCR).”

11.1.4 Debug Port Hard or Soft Reset

When the development port receives a hard or soft reset request from a development tool, an internal hard or soft reset sequence is generated. The development tool must reconfigure the debug port following a reset event. See **Section 44.3.2.1.2, “Development Serial Data In (DSDI)**.”

11.1.5 JTAG Reset

When the JTAG logic asserts the JTAG reset signal, an internal soft reset sequence is generated.

11.1.6 Power-On and Hard Reset Sequence

Figure 11-1 shows the reset sequence following a power-on or internal or external hard reset event.

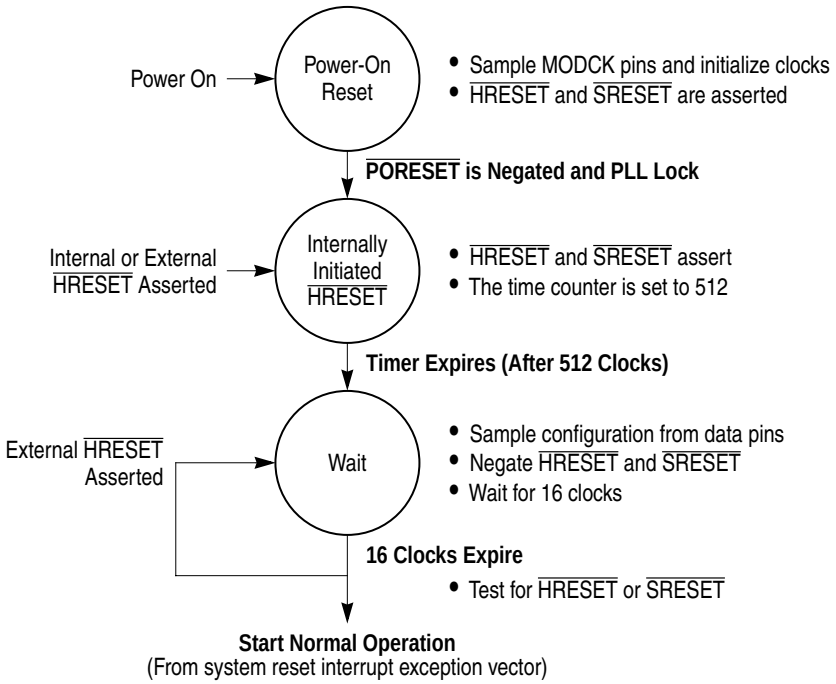


Figure 11-1. Power-On and Hard Reset Sequence

11.1.7 External Soft Reset

When an external $\overline{\text{SRESET}}$ is asserted, the core starts driving the $\overline{\text{SRESET}}$ signal. After 512 clock cycles the debug port configuration is sampled from the DSDI and DSCK signals and the core stops driving the $\overline{\text{SRESET}}$ signal. Once the core negates $\overline{\text{SRESET}}$ 16 clock cycles must elapse before the external soft reset signal is again sampled.

The soft reset ($\overline{\text{SRESET}}$) signal is also a bidirectional, active low, open-collector I/O signal. The MPC855T can detect an external assertion of $\overline{\text{SRESET}}$ only if it occurs while the MPC855T is not internally asserting $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$.

11.1.8 Internal Soft Reset

The JTAG and debug ports can initiate an internal soft reset, resulting in the assertion of the $\overline{\text{SRESET}}$ signal. After 512 cycles, the core negates $\overline{\text{SRESET}}$ and the debug port configuration is sampled from the DSDI and DSCK signals. Once the core negates $\overline{\text{SRESET}}$, 16 clock cycles must elapse before the external soft reset signal is sampled.

11.1.9 Soft Reset Sequence

Figure 11-2 shows the reset sequence following an internal or external soft reset event.

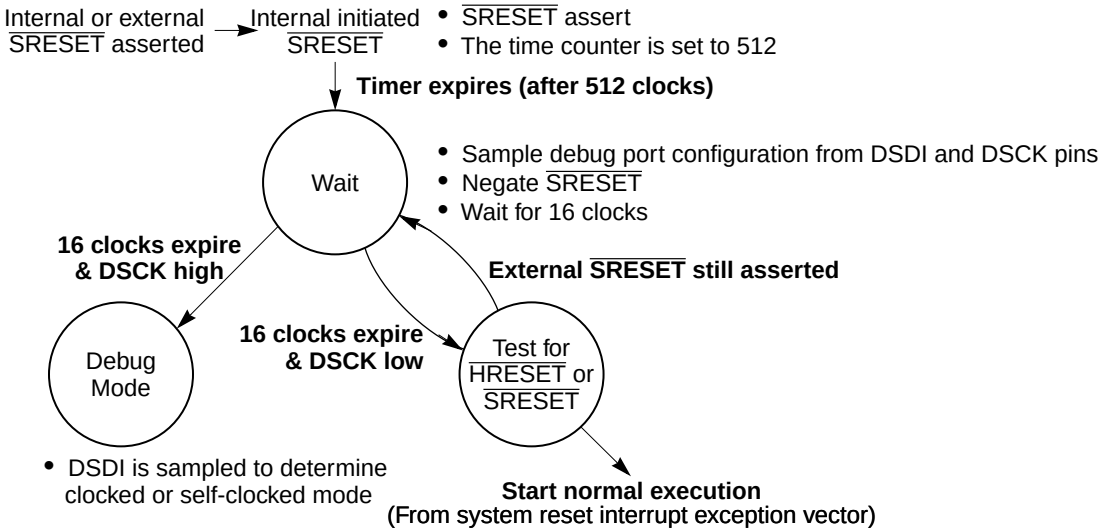


Figure 11-2. Soft Reset Sequence

11.2 Reset Status Register (RSR)

The 32-bit reset status register (RSR) is powered by the keep alive power supply. It is memory-mapped into the MPC855T system interface unit register map and receives its default reset values at power-on reset. This register is also effected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

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	—							
Reset	1100_0000_0000_0000															
R/W	R/W															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—															
reset	0000_0000_0000_0000															
r/w	R/W															

Figure 11-3. Reset Status Register (RSR)

The RSR bits are described in Table 11-2 Note that the bits in this register (except those that are reserved) are cleared by writing ones; writing zeros has no effect.

Table 11-2. Reset Status Register Bit Settings

Bits	Name	Description
0	EHRS	External hard reset status. Set by a power-on reset. When an external hard reset event is detected, EHRS is set and remains set until software clears it. 0 No external hard reset event occurred. 1 An external hard reset event occurred.
1	ESRS	External soft reset status. Set by a power-on reset. When an external soft reset event is detected, ESRS is set and remains set until software clears it. 0 No external soft reset event occurred. 1 An external soft reset event occurred.
2	LLRS	Loss-of-lock reset status. Cleared by a power-on reset. When a loss-of-lock event (enabled by PLPRCR[LOLRE]) is detected, LLRS is set and remains set until software clears it. LLRS is affected only by an unintentional loss of lock due to a hardware-related issue. A software-initiated loss of lock, such as changing PLPRCR[MF] or entering deep-sleep or power-down mode, does not affect LLRS. 0 No enabled loss-of-lock reset event occurred. 1 An enabled loss-of-lock reset event occurred.
3	SWRS	Software watchdog reset status. Cleared by a power-on reset. When a software watchdog expire event occurs, SWRS is set and remains set until software clears it. 0 No software watchdog reset event occurred. 1 A software watchdog reset event occurred.
4	CSRS	Check stop reset Status. Cleared by a power-on reset. When the core enters the checkstop state and the checkstop reset is enabled by PLPRCR[CSR], CSRS is set and remains set until software clears it. 0 No enabled checkstop reset event occurred. 1 An enabled checkstop reset event occurred.
5	DBHRS	Debug port hard reset status. Cleared by a power-on reset. When the debug port hard reset request is set, DBHRS is set and remains set until software clears it. 0 No debug port hard reset request occurred. 1 A debug port hard reset request occurred.

Table 11-2. Reset Status Register Bit Settings (continued)

Bits	Name	Description
6	DBSRS	Debug port soft reset status. Cleared by a power-on reset. When the debug port soft reset request is set, DBSRS is set and remains set until software clears it. 0 No debug port soft reset request occurred. 1 A debug port soft reset request occurred.
7	JTRS	JTAG reset status. Cleared by a power-on reset. When the JTAG reset request is set, this bit is set and remains set until software clears it. 0 No JTAG reset event occurred. 1 A JTAG reset event occurred.
8–31	—	Reserved and should be cleared.

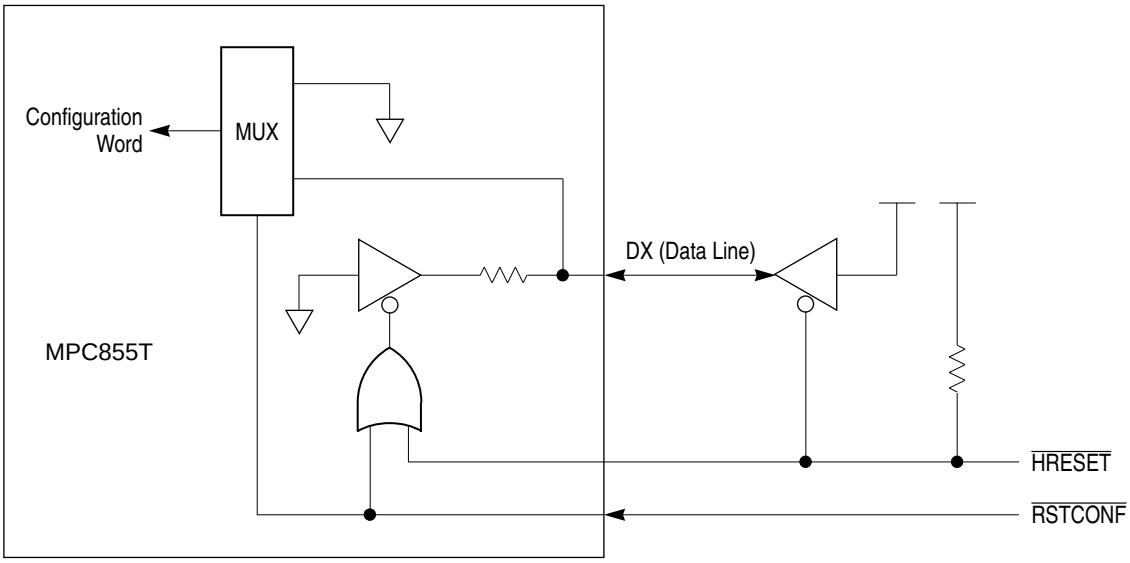
11.3 MPC855T Reset Configuration

When a hard reset event occurs, the MPC855T reconfigures both its internal hardware and the development port. A soft reset is used to reconfigure the development port without changing the MPC855T's internal machine state. The following sections describe the configuration of the MPC855T using hard and soft reset events.

11.3.1 Hard Reset

When a hard reset event occurs, the MPC855T determines its initial mode of operation by sampling the values present on the data bus (D[0–31]) or from an internal default constant (D[0–31] = 0x00000000). If the $\overline{\text{RSTCONF}}$ signal is asserted at sampling time, the configuration is sampled from the data bus. If the $\overline{\text{RSTCONF}}$ signal is negated the internal default value is selected. While $\overline{\text{HRESET}}$ and $\overline{\text{RSTCONF}}$ are asserted, the MPC855T weakly pulls the data bus low, and the desired configuration is selected by driving the appropriate bits high as shown in Figure 11-4.

Figure 11-4 shows a typical data bus configuration input circuit.



NOTE: The value of the internal pulldown resistor is not specified or guaranteed.

Figure 11-4. Data Bus Configuration Input Circuit

The configuration of the MPC855T following the assertion of $\overline{\text{PORESET}}$ is shown in Figure 11-5 through Figure 11-7 While the $\overline{\text{PORESET}}$ input signal is being asserted, the core assumes the default reset configuration (0x0000_0000). When $\overline{\text{PORESET}}$ is negated or the CLKOUT signal begins oscillation, the hardware configuration is sampled from the data bus every nine clock cycles on the rising edge of 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. Refer to Section 11.3.2, “Soft Reset,” for more information.

Figure 11-5 shows a reset operation with a short $\overline{\text{PORESET}}$ signal assertion. Note that the configuration of the MPC855T is determined from the signal levels driven on the D[0–31] signals following the assertion of $\overline{\text{RSTCONF}}$ and the negation of $\overline{\text{HRESET}}$.

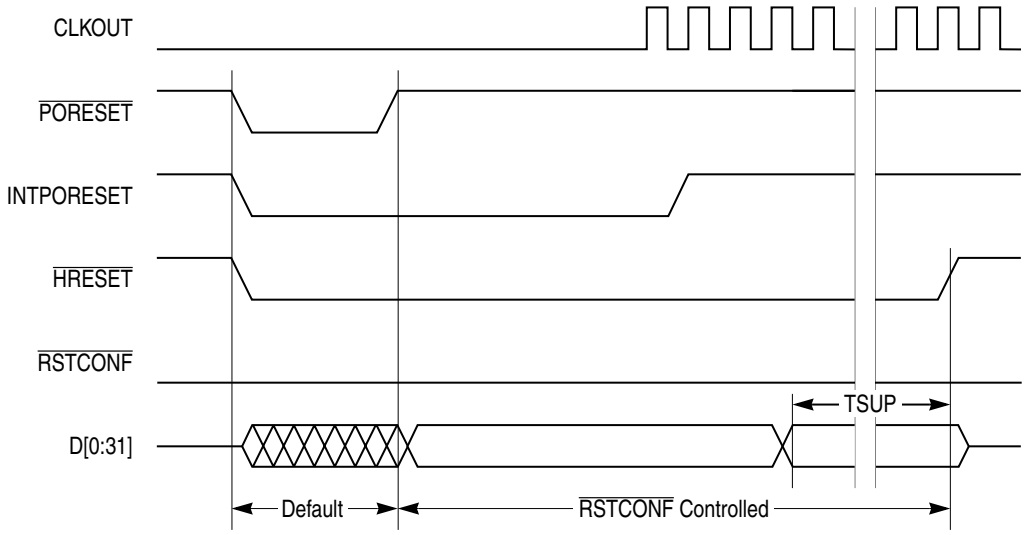


Figure 11-5. Reset Configuration Sampling for Short PORESET Assertion

Figure 11-6 shows a reset operation with a long PORESET signal assertion.

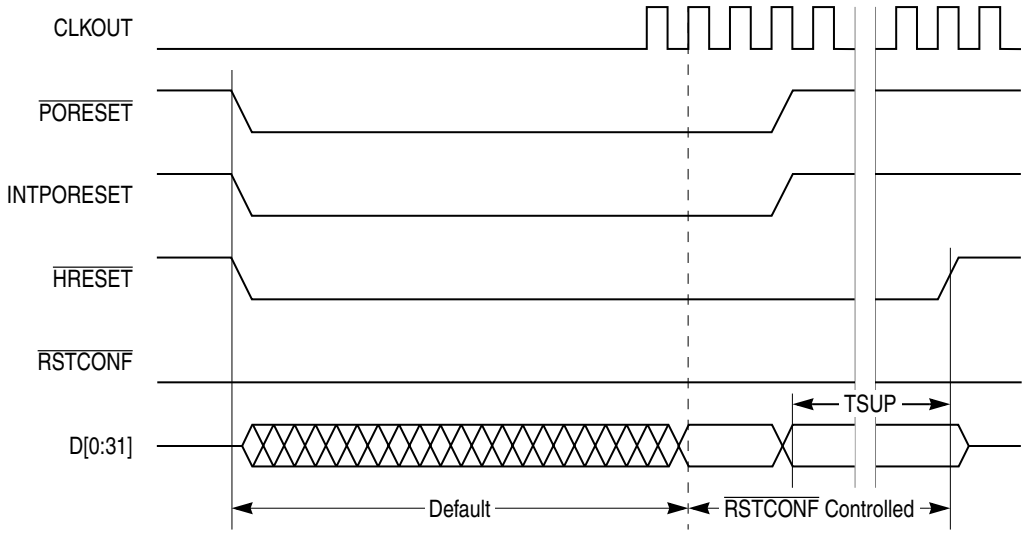


Figure 11-6. Reset Configuration Sampling for Long PORESET Assertion

Figure 11-7 shows the configuration data sampling timing relative to HRESET and CLKOUT.

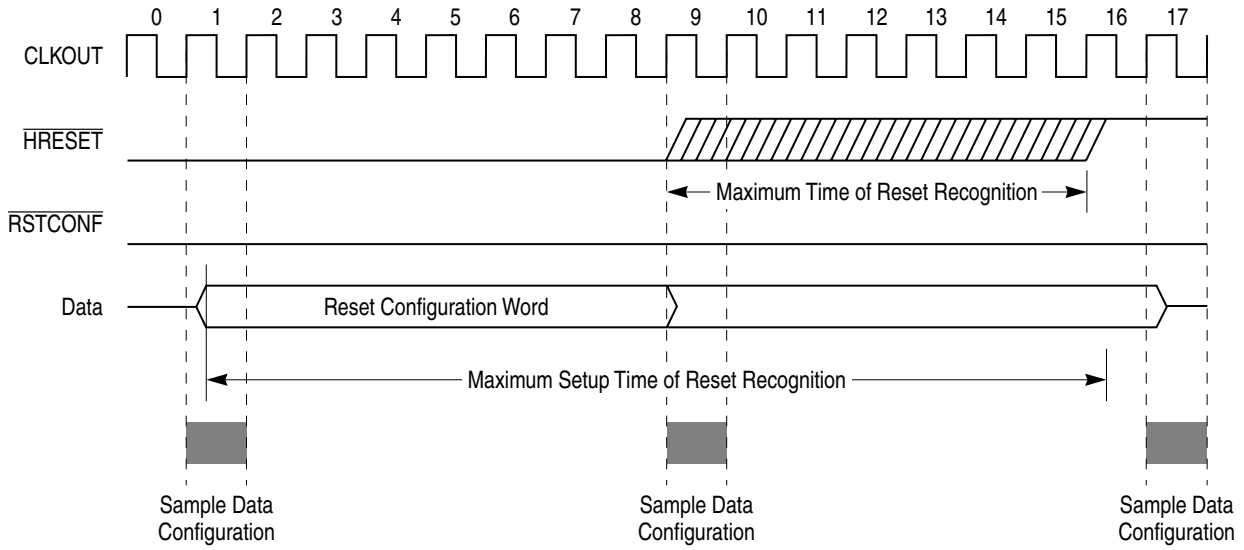


Figure 11-7. Reset Configuration Sampling Timing Requirements

11.3.1.1 Hard Reset Configuration Word

The hard reset configuration word is sampled from the data bus. These bits determine the default values of the corresponding bits in the SIUMCR, IMMR, and MSR.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	EARB	IIP	BBE	BDIS	BPS		—	ISB		DBGC		DBPC		EBDF		CLES
Default	0000_0000_0000_0000															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—															
Default	0000_0000_0000_0000															

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

Figure 11-8. Hard Reset Configuration Word

Table 11-3 describes hard reset configuration word fields.

Table 11-3. Hard Reset Configuration Word Field Descriptions

Bits	Name	Description
0	EARB	External arbitration. If this bit is set, external arbitration is assumed. If it is cleared, internal arbitration is performed. See Section 10.4.2, “SIU Module Configuration Register (SIUMCR).”
1	IIP	Initial interrupt prefix. Defines the initial value of the MSR[IP] which defines the interrupt table location. If IIP is cleared (default), the MSR[IP] initial value is one; if it is set, the MSR[IP] initial value is zero. See Section 4.1.2.3.1, “Machine State Register (MSR).”
2	BBE	Boot Burst Enable 0 The boot device does not support bursting. 1 The boot device does support bursting.

Table 11-3. Hard Reset Configuration Word Field Descriptions (continued)

Bits	Name	Description
3	BDIS	Boot disable. If BDIS is set, memory bank 0 is invalid; that is, BR0[V] is cleared. (See Section 15.4.1, “Base Registers (BRx).”) 0 The memory controller is activated after reset so that it matches all addresses. 1 The memory controller is cleared after reset but is not activated.
4–5	BPS	Boot port size. Defines the port size of the boot device as shown in the following chart. 00 32-bit port size. 01 8-bit port size. 10 16-bit port size. 11 Reserved.
6	—	Reserved for future use and should be allowed to float.
7–8	ISB	Initial internal space base select. 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.
9–10	DBGC	Debug pin configuration. Selects the signal function of the following pins:

Pin	DBGC = 00	DBGC = 01	DBGC = 10	DBGC = 11
IP_B[0–1]/IWP[0–1]/VFLS[0–1]	IP_B[0–1]	IWP[0–1]	Reserved	VFLS[0–1]
IP_B3/IWP2/VF2	IP_B3	IWP2		VF2
IP_B4/LWP0/VF0	IP_B4	LWP0		VF0
IP_B5/LWP1/VF1	IP_B5	LWP1		VF1
OP2/MODCK1/$\overline{\text{STS}}$	OP2	$\overline{\text{STS}}$		$\overline{\text{STS}}$
ALE_B/DSCK/AT1	ALE_B	AT1		AT1
IP_B2/$\overline{\text{IOIS16\_B}}$/AT2	IP_B2	AT2		AT2
IP_B6/DSDI/AT0	IP_B6	AT0		AT0
IP_B7/$\overline{\text{PTR}}$/AT3	IP_B7	AT3		AT3
OP3/MODCK2/DSDO	OP3	OP3		OP3

11–12	DBPC	Debug port pins configuration. Selects the signal function for the following development port pins:

Pin	DBPC = 00	DBPC = 01	DBPC = 10	DBPC = 11
ALE_B/DSCK/AT1	Defined by DBGC. Note that if DBPC = 11, DBPC overrides DBGC.		Reserved	DSCK
IP_B6/DSDI/AT0				DSDI
OP3/MODCK2/DSDO				DSDO
IP_B7/$\overline{\text{PTR}}$/AT3				$\overline{\text{PTR}}$
TCK/DSCK	DSCK	TCK		TCK
TDI/DSDI	DSDI	TDI		TDI
TDO/DSDO	DSDO	TDO		TDO

Table 11-3. Hard Reset Configuration Word Field Descriptions (continued)

Bits	Name	Description
13–14	EBDF	External bus division factor. Defines 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. Refer to Chapter 14, “Clocks and Power Control” for additional information. 00 Full speed bus 01 Half speed bus 10 Reserved 11 Reserved
15	CLES	Core Little Endian Swap. Defines core access operation following reset. 0 Big Endian 1 Little Endian

11.3.2 Soft Reset

When a soft reset event occurs, the MPC855T reconfigures the development port. See Section 44.3.1.2, “Entering Debug Mode,” and Section 44.3.2.3.3, “Selection of Development Port Clock Mode.”

11.4 $\overline{\text{TRST}}$ and Power Mode Considerations

Note the following when connecting the $\overline{\text{TRST}}$ (test reset) signal:

- If both low power mode and the TAP are never used, connect $\overline{\text{TRST}}$ to ground.
- If low power mode (or the TAP) is used, connect $\overline{\text{TRST}}$ to $\overline{\text{PORESET}}$.
- If power down mode (the lowest power mode, where V_{DDH} is disabled) is used, connect $\overline{\text{TRST}}$ to $\overline{\text{PORESET}}$ through a diode (anode to $\overline{\text{TRST}}$, cathode to $\overline{\text{PORESET}}$).

See also Section 45.6, “Recommended TAP Configuration.”

Part IV
Hardware Interface

Intended Audience

Part IV is intended for system designers who need to understand how each MPC855T signal works and how those signals interact.

Contents

Part IV describes external signals, clocking, memory control, and power management of the MPC855T. It contains the following chapters:

- Chapter 12, “External Signals,” provides a detailed description of the external signals that comprise the MPC855T external interface.
- Chapter 13, “External Bus Interface,” describes interactions among signals described in the previous chapter, including numerous examples and timing diagrams.
- Chapter 14, “Clocks and Power Control,” describes on-chip and external devices, including the phase-locked loop circuitry and frequency dividers that generate programmable clock timing for baud-rate generators, timers, and a variety of low-power mode options.
- Chapter 15, “Memory Controller,” describes the memory controller, which controlling a maximum of eight memory banks shared between a general-purpose chip-select machine (GPCM) and a pair of user-programmable machines (UPMs).
- Chapter 16, “PCMCIA Interface,” describes the PCMCIA host adapter module, which provides all control logic for a PCMCIA socket interface and requires only additional external analog power switching logic and buffering.

Suggested Reading

This section lists additional reading that provides background for the information in this manual.

MPC8xx Documentation

Supporting documentation for the MPC855T can be accessed through the world-wide web at http://www.motorola.com. This documentation includes technical specifications, reference materials, and detailed applications notes.

Conventions

This document uses the following notational conventions:

Bold	Bold entries in figures and tables showing registers and parameter RAM should be initialized by the user.
mnemonics	Instruction mnemonics are shown in lowercase bold.
italics	Italics indicate variable command parameters, for example, **bcctr***x*. Book titles in text are set in italics.
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote binary number
REG[FIELD]	Abbreviations or acronyms for registers or buffer descriptors are shown in uppercase text. Specific bits, fields, or numerical ranges appear in brackets. For example, MSR[LE] refers to the little-endian mode enable bit in the machine state register.
x	In certain contexts, such as in a signal encoding or a bit field, indicates a don’t care.
n	Indicates an undefined numerical value
¬	NOT logical operator
&	AND logical operator
|	OR logical operator

Acronyms and Abbreviations

Table ii contains acronyms and abbreviations used in this document. Note that the meanings for some acronyms (such as SDR1 and DSISR) are historical, and the words for which an acronym stands may not be intuitively obvious.

Table ii. Acronyms and Abbreviated Terms

Term	Meaning
BD	Buffer descriptor
BIST	Built-in self test
BRI	Basic rate interface
BUID	Bus unit ID
CAM	Content-addressable memory
CPM	Communications processor module
CRC	Cyclic redundancy check
DMA	Direct memory access
DPLL	Digital phase-locked loop
DRAM	Dynamic random access memory
DSISR	Register used for determining the source of a DSI exception
EA	Effective address
EEST	Enhanced Ethernet serial transceiver
GCI	General circuit interface
GPCM	General-purpose chip-select machine
HDLC	High-level data link control
I^2C	Inter-integrated circuit
IDL	Inter-chip digital link
IEEE	Institute of Electrical and Electronics Engineers
IrDA	Infrared Data Association
ISDN	Integrated services digital network
JTAG	Joint Test Action Group
LIFO	Last-in-first-out
LRU	Least recently used
LSB	Least-significant byte
lsb	Least-significant bit
LSU	Load/store unit
MAC	Multiply accumulate
MMU	Memory management unit

Table ii. Acronyms and Abbreviated Terms (continued)

Term	Meaning
MSB	Most-significant byte
msb	Most-significant bit
MSR	Machine state register
NMSI	Nonmultiplexed serial interface
OSI	Open systems interconnection
PCI	Peripheral component interconnect
PCMCIA	Personal Computer Memory Card International Association
PRI	Primary rate interface
Rx	Receive
SCC	Serial communications controller
SCP	Serial control port
SDLC	Synchronous data link control
SDMA	Serial DMA
SI	Serial interface
SIU	System interface unit
SMC	Serial management controller
SNA	Systems network architecture.
SPI	Serial peripheral interface
SPR	Special-purpose register
SRAM	Static random access memory
TDM	Time-division multiplexed
TLB	Translation lookaside buffer
TSA	Time-slot assigner
Tx	Transmit
UART	Universal asynchronous receiver/transmitter
UISA	User instruction set architecture
UPM	User-programmable machine
USART	Universal synchronous/asynchronous receiver/transmitter

Chapter 12
External Signals

This chapter contains descriptions of the MPC855T input and output signals, showing multiplexing, pin assignments, and reset values.

Figure 12-1 shows the signals grouped by function.

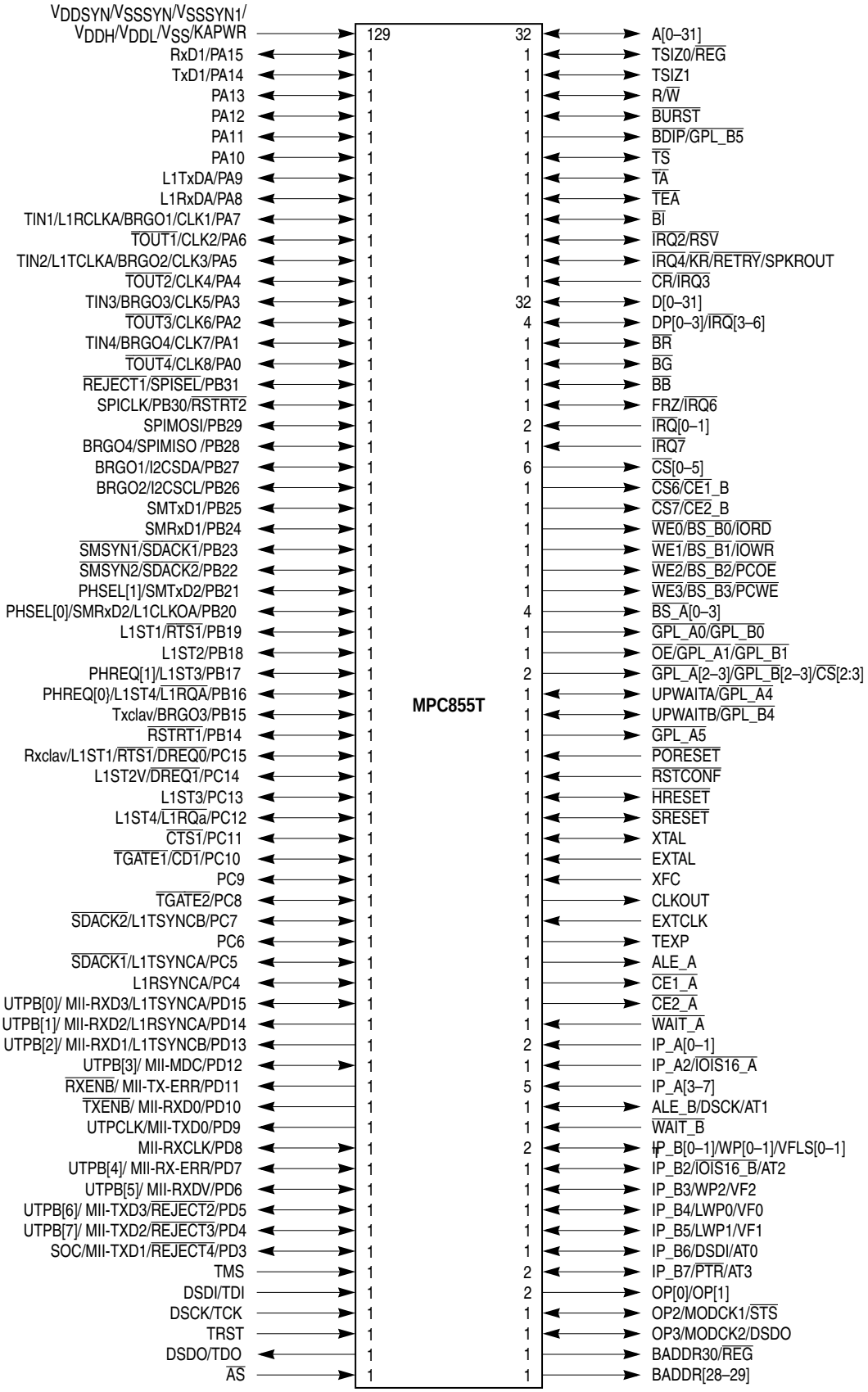


Figure 12-1. MPC855T External Signals

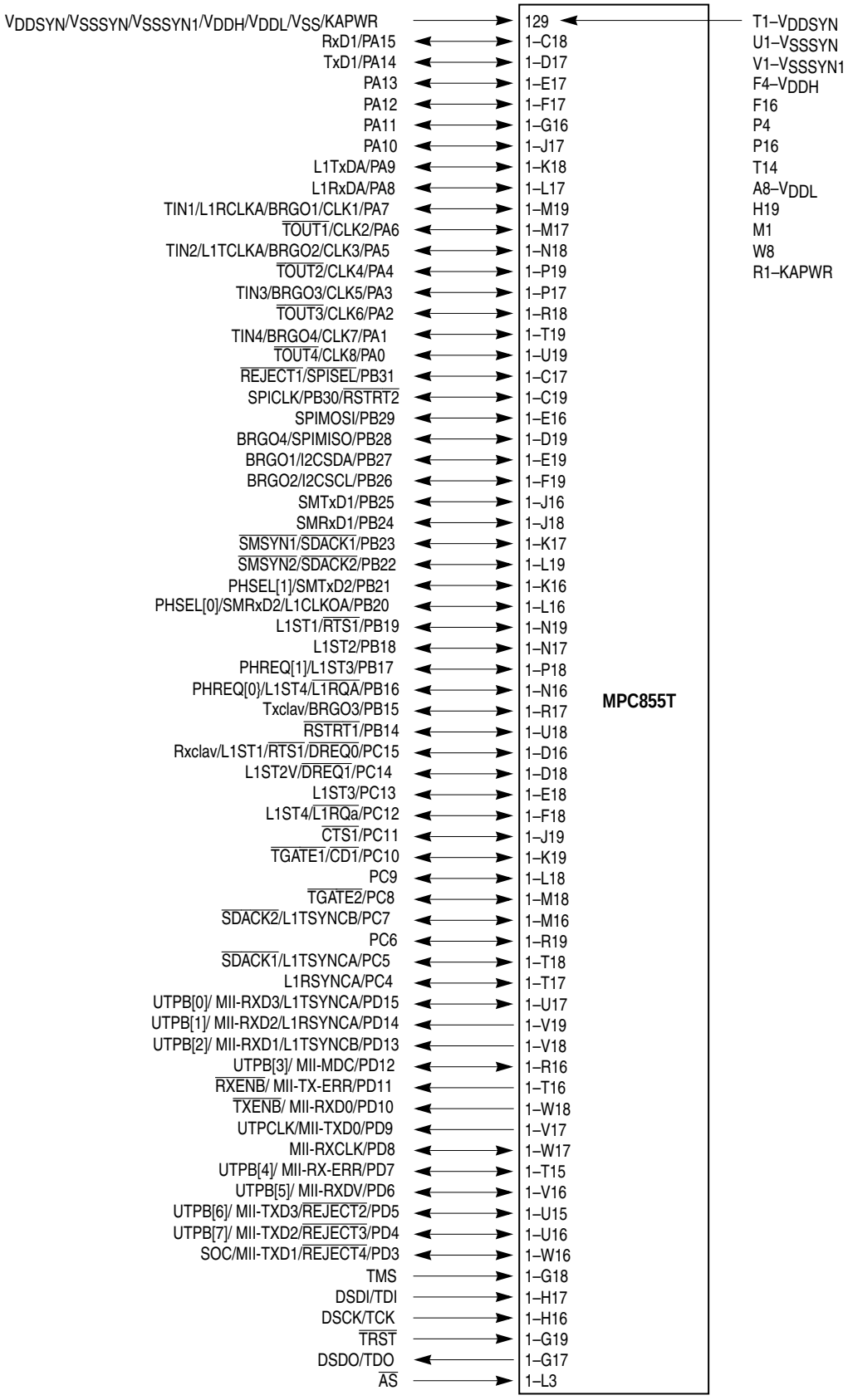


Figure 12-2. Signals and Pin Numbers (Part 1)

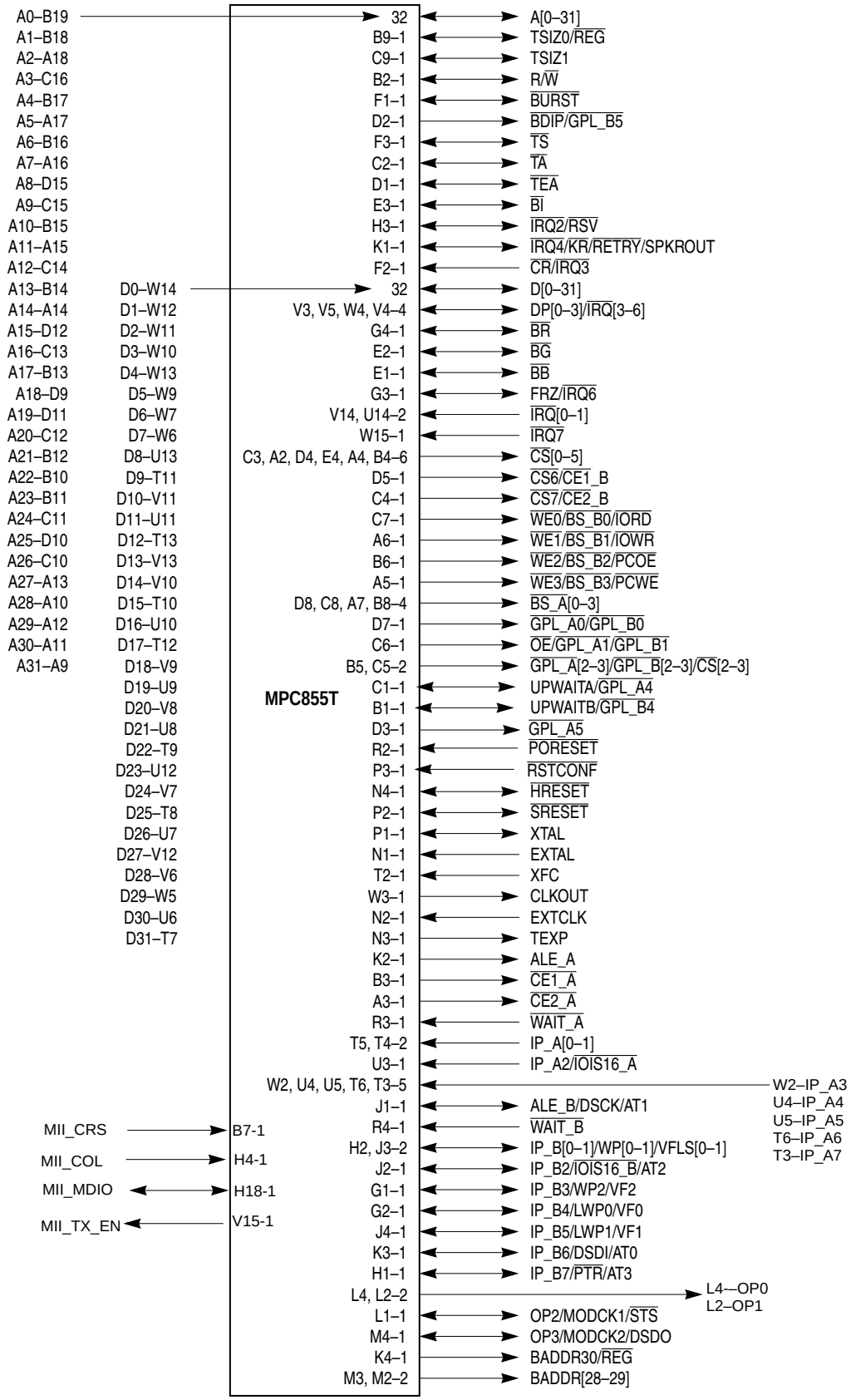


Figure 12-3. Signals and Pin Numbers (Part 2)

12.1 System Bus Signals

The MPC855T system bus consists of all signals that interface with the external bus. Many of these signals perform different functions, depending on how the user assigns them. The input and output signals in Table 12-1 are identified by their abbreviation.

Table 12-1. Signal Descriptions

Name	Hard Reset	Number	Type	Description
A[0–31]	Hi-Z	See Figure	Bidirectional Three-state	Address Bus—Provides the address for the current bus cycle. A0 is the msb. The bus is output when an internal master starts a transaction on the external bus. The bus is input when an external master starts a transaction on the bus.
TSIZ0 REG	Hi-Z	B9	Bidirectional Three-state	Transfer Size 0—When accessing a slave in the external bus, used (together with TSIZ1) by the bus master to indicate the number of operand bytes waiting to be transferred in the current bus cycle. TSIZ0 is an input when an external master starts a bus transaction. Register—When an internal master initiates an access to a slave controlled by the PCMCIA interface, $\overline{\text{REG}}$ is output to indicate which space in the PCMCIA card is accessed.
TSIZ1	Hi-Z	C9	Bidirectional Three-state	Transfer Size 1—Used (with TSIZ0) by the bus master to indicate the number of operand bytes waiting to be transferred in the current bus cycle. The MPC855T drives TSIZ1 when it is bus master. TSIZ1 is input when an external master starts a bus transaction.
RD/$\overline{\text{WR}}$	Hi-Z	B2	Bidirectional Three-state	Read/Write—Driven by a bus master to indicate the direction of the data transfer. A logic one indicates a read from a slave device and a logic zero indicates a write to a slave device. The MPC855T drives this signal when it is bus master. Input when an external master initiates a transaction on the bus.
$\overline{\text{BURST}}$	Hi-Z	F1	Bidirectional Three-state	Burst Transaction—Driven by the bus master to indicate that the current initiated transfer is a burst. The MPC855T drives this signal when it is bus master. This signal is input when an external master initiates a transaction on the bus.
$\overline{\text{BDIP}}$ $\overline{\text{GPL\_B5}}$	See Table 12-2	D2	Output	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. $\overline{\text{BDIP}}$ is negated before the expected last data beat of the burst transfer. General-Purpose Line B5—Used by the memory controller when UPMB takes control of the slave access.

Table 12-1. Signal Descriptions (continued)

Name	Hard Reset	Number	Type	Description
$\overline{TS}$	Hi-Z	F3	Bidirectional Active Pull-up	Transfer Start—Asserted by a bus master to indicate the start of a bus cycle that transfers data to or from a slave device. Driven by the master only when it has gained the ownership of the bus. Every master should negate this signal before the bus relinquish. $\overline{TS}$ requires the use of an external pull-up resistor. The MPC855T samples $\overline{TS}$ when it is not the external bus master to allow the memory controller/PCMCIA interface to control the accessed slave device. It indicates that an external synchronous master initiated a transaction.
$\overline{TA}$	Hi-Z	C2	Bidirectional Active Pull-up	Transfer Acknowledge—Indicates that the slave device addressed in the current transaction accepted data sent by the master (write) or has driven the data bus with valid data (read). This is an output when the PCMCIA interface or memory controller controls the transaction. The only exception occurs when the memory controller controls the slave access by means of the GPCM and the corresponding option register is instructed to wait for an external assertion of $\overline{TA}$. Every slave device should negate $\overline{TA}$ after a transaction ends and immediately three-state it to avoid bus contention if a new transfer is initiated addressing other slave devices. $\overline{TA}$ requires the use of an external pull-up resistor.
$\overline{TEA}$	Hi-Z	D1	Open-drain	Transfer Error Acknowledge—Indicates that a bus error occurred in the current transaction. The MPC855T asserts $\overline{TEA}$ when the bus monitor does not detect a bus cycle termination within a reasonable amount of time. Asserting $\overline{TEA}$ terminates the bus cycle, thus ignoring the state of $\overline{TA}$. $\overline{TEA}$ requires the use of an external pull-up resistor.
$\overline{BI}$	Hi-Z	E3	Bidirectional Active Pull-up	Burst Inhibit—Indicates that the slave device addressed in the current burst transaction cannot support burst transfers. It acts as an output when the PCMCIA interface or the memory controller takes control of the transaction. $\overline{BI}$ requires the use of an external pull-up resistor.
$\overline{RSV}$ $\overline{IRQ2}$	See Table 12-2	H3	Bidirectional Three-state	Reservation—The MPC855T outputs this three-state signal in conjunction with the address bus to indicate that the core initiated a transfer as a result of a **stwcx**. or **lwarx**. Interrupt Request 2—One of eight external inputs that can request (by means of the internal interrupt controller) a service routine from the core.

Table 12-1. Signal Descriptions (continued)

Name	Hard Reset	Number	Type	Description
KR/RETRY IRQ4 SPKROUT	See Table 12-2	K1	Bidirectional Three-state	Kill Reservation—Input used as a part of the memory reservation protocol, when the MPC855T initiated a transaction as the result of a **stwcx**. instruction. Retry—Input used by a slave device to indicate it cannot accept the transaction. The MPC855T must relinquish mastership and reinitiate the transaction after winning in the bus arbitration. Interrupt Request 4. One of eight external inputs that can request (by means of the internal interrupt controller) a service routine from the core. Note that the interrupt request signal that is sent to the interrupt controller is the logical AND of this line (if defined as IRQ4) and DP1/IRQ4 (if defined as IRQ4). SPKROUT—Digital audio wave form output to be driven to the system speaker.
CR IRQ3	Hi-Z	F2	Input	Cancel Reservation—Input used as a part of the storage reservation protocol. Interrupt Request 3—One of eight external inputs that can request (by means of the internal interrupt controller) a service routine from the core. Note that the interrupt request signal sent to the interrupt controller is the logical AND of CR/IRQ3 (if defined as IRQ3) and DP0/IRQ3 if defined as IRQ3.
D[0–31]	Hi-Z[1]	See Figure	Bidirectional Three-state	Data Bus—Bidirectional three-state bus, provides the general-purpose data path between the MPC855T and all other devices. The 32-bit data path can be dynamically sized to support 8-, 16-, or 32-bit transfers. D0 is the msb of the data bus.
DP0 IRQ3	Hi-Z	V3	Bidirectional Three-state	Data Parity 0—Provides parity generation and checking for D[0–7] for transfers to a slave device initiated by the MPC855T. 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. Parity generation and checking is not supported for external masters. Interrupt Request 3—One of eight external inputs that can request (by means of the internal interrupt controller) a service routine from the core. Note that the interrupt request signal sent to the interrupt controller is the logical AND of DP0/IRQ3 (if defined as IRQ3) and CR/IRQ3 (if defined as IRQ3).

Table 12-1. Signal Descriptions (continued)

Name	Hard Reset	Number	Type	Description
DP1 IRQ4	Hi-Z	V5	Bidirectional Three-state	Data Parity 1—Provides parity generation and checking for D[8–15] for transfers to a slave device initiated by the MPC855T. 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. Parity generation and checking is not supported for external masters. Interrupt Request 4—One of eight external inputs that can request (by means of the internal interrupt controller) a service routine from the core. Note that the interrupt request signal sent to the interrupt controller is the logical AND of this line (if defined as IRQ4) and KR/IRQ4/SPKROUT (if defined as IRQ4).
DP2 IRQ5	Hi-Z	W4	Bidirectional Three-state	Data Parity 2—Provides parity generation and checking for D[16–23] for transfers to a slave device initiated by the MPC855T. 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. Parity generation and checking is not supported for external masters. Interrupt Request 5—One of eight external inputs that can request (by means of the internal interrupt controller) a service routine from the core.
DP3 IRQ6	Hi-Z	V4	Bidirectional Three-state	Data Parity 3—Provides parity generation and checking for D[16–23] for transfers to a slave device initiated by the MPC855T. 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. Parity generation and checking is not supported for external masters. Interrupt Request 6—One of eight external inputs that can request (by means of the internal interrupt controller) a service routine from the core. Note that the interrupt request signal sent to the interrupt controller is the logical AND of this line (if defined as IRQ6) and the FRZ/IRQ6 (if defined as IRQ6).
BR	Hi-Z	G4	Bidirectional	Bus Request—Asserted low when a possible master is requesting ownership of the bus. When the MPC855T is configured to work with the internal arbiter, this signal is configured as an input. When the MPC855T is configured to work with an external arbiter, this signal is configured as an output.

Table 12-1. Signal Descriptions (continued)

Name	Hard Reset	Number	Type	Description
$\overline{BG}$	Hi-Z	E2	Bidirectional	Bus Grant—Asserted low when the arbiter of the external bus grants the bus to a specific device. When the MPC855T is configured to work with the internal arbiter, $\overline{BG}$ is configured as an output and asserted every time the external master asserts $\overline{BR}$ and its priority request is higher than any internal sources requiring a bus transfer. However, when the MPC855T is configured to work with an external arbiter, $\overline{BG}$ is an input.
$\overline{BB}$	Hi-Z	E1	Bidirectional Active Pull-up	Bus Busy—Asserted low by a master to show that it owns the bus. The MPC855T asserts $\overline{BB}$ after the arbiter grants it bus ownership and $\overline{BB}$ is negated.
FRZ $\overline{IRQ6}$	See Table 12-2	G3	Bidirectional	Freeze—Output asserted to indicate that the core is in debug mode. Interrupt Request 6—One of eight external inputs that can request (by means of the internal interrupt controller) a service routine from the core. Note that the interrupt request signal sent to the interrupt controller is the logical AND of FRZ/$\overline{IRQ6}$ (if defined as $\overline{IRQ6}$) and DP3/$\overline{IRQ6}$ (if defined as $\overline{IRQ6}$).
$\overline{IRQ0}$	Hi-Z	V14	Input	Interrupt Request 0—One of eight external inputs that can request (by means of the internal interrupt controller) a service routine from the core.
$\overline{IRQ1}$	Hi-Z	U14	Input	Interrupt Request 1—One of eight external inputs that can request (by means of the internal interrupt controller) a service routine from the core.
$\overline{IRQ7}$	Hi-Z	W15	Input	Interrupt Request 7—One of eight external inputs that can request (by means of the internal interrupt controller) a service routine from the core.
$\overline{CS}$[0–5]	High	C3, A2, D4, E4, A4, B4	Output	Chip Select—These outputs enable peripheral or memory devices at programmed addresses if they are appropriately defined. $\overline{CS0}$ can be configured to be the global chip-select for the boot device.
$\overline{CS6}$ $\overline{CE1\_B}$	High	D5	Output	Chip Select 6—This output enables a peripheral or memory device at a programmed address if defined appropriately in the BR6 and OR6 in the memory controller. Card Enable 1 Slot B—This output enables even byte transfers when accesses to the PCMCIA Slot B are handled under the control of the PCMCIA interface.
$\overline{CS7}$ $\overline{CE2\_B}$	High	C4	Output	Chip Select 7—This output enables a peripheral or memory device at a programmed address if defined appropriately in the BR7 and OR7 in the memory controller. Card Enable 2 Slot B—This output enables odd byte transfers when accesses to the PCMCIA Slot B are handled under the control of the PCMCIA interface.

Table 12-1. Signal Descriptions (continued)

Name	Hard Reset	Number	Type	Description
WE0 BS_B0 IORD	High	C7	Output	Write Enable 0—Output asserted when a write access to an external slave controlled by the GPCM is initiated by the MPC855T. WE0 is asserted if D[0–7] contains valid data to be stored by the slave device. Byte Select 0 on UPMB—Output asserted under control of the UPMB, as programmed by the user. In a read or write transfer, the line is only asserted if D[0–7] contains valid data. IO Device Read—Output asserted when the MPC855T starts a read access to a region controlled by the PCMCIA interface. Asserted only for accesses to a PC card I/O space.
WE1 BS_B1 IOWR	High	A6	Output	Write Enable 1—Output asserted when the MPC855T initiates a write access to an external slave controlled by the GPCM. WE1 is asserted if D[8–15] contains valid data to be stored by the slave device. Byte Select 1 on UPMB—Output asserted under control of the UPMB, as programmed by the user. In a read or write transfer, the line is only asserted if D[8–15] contains valid data. I/O Device Write—This output is asserted when the MPC855T initiates a write access to a region controlled by the PCMCIA interface. IOWR is asserted only if the access is to a PC card I/O space.
WE2 BS_B2 PCOE	High	B6	Output	Write Enable 2—Output asserted when the MPC855T starts a write access to an external slave controlled by the GPCM. WE2 is asserted if D[16–23] contains valid data to be stored by the slave device. Byte Select 2 on UPMB—Output asserted under control of the UPMB, as programmed by the user. In a read or write transfer, BS_B2 is asserted only D[16–23] contains valid data. PCMCIA Output Enable—Output asserted when the MPC855T initiates a read access to a memory region under the control of the PCMCIA interface.
WE3 BS_B3 PCWE	High	A5	Output	Write Enable 3—Output asserted when the MPC855T initiates a write access to an external slave controlled by the GPCM. WE3 is asserted if D[24–31] contains valid data to be stored by the slave device. Byte Select 3 on UPMB—Output asserted under control of the UPMB, as programmed by the user. In a read or write transfer, BS_B3 is asserted only if D[24–31] contains valid data. PCMCIA Write Enable—Output asserted when the MPC855T initiates a write access to a memory region under control of the PCMCIA interface.
BS_A[0–3]	High	D8, C8, A7, B8	Output	Byte Select 0 to 3 on UPMA—Outputs asserted under requirement of the UPMA, as programmed by the user. For read or writes, asserted only if their corresponding data lanes contain valid data: BS_A0 for D[0–7], BS_A1 for D[8–15], BS_A2 for D[16–23], BS_A3 for D[24–31]

Table 12-1. Signal Descriptions (continued)

Name	Hard Reset	Number	Type	Description
GPL_A0 GPL_B0	High	D7	Output	General-Purpose Line 0 on UPMA—This output reflects the value specified in the UPMA when an external transfer to a slave is controlled by the UPMA. General-Purpose Line 0 on UPMB—This output reflects the value specified in the UPMB when an external transfer to a slave is controlled by the UPMB.
OE GPL_A1 GPL_B1	High	C6	Output	Output Enable—Output asserted when the MPC855T initiates a read access to an external slave controlled by the GPCM. General-Purpose Line 1on UPMA—This output reflects the value specified in the UPMA when an external transfer to a slave is controlled by UPMA. General-Purpose Line 1 on UPMB—This output reflects the value specified in the UPMB when an external transfer to a slave is controlled by UPMB.
GPL_A[2–3] GPL_B[2–3] CS[2–3]	High	B5, C5	Output	General-Purpose Line 2 and 3 on UPMA—These outputs reflect the value specified in the UPMA when an external transfer to a slave is controlled by UPMA. General-Purpose Line 2 and 3 on UPMB—These outputs reflect the value specified in the UPMB when an external transfer to a slave is controlled by UPMB. Chip Select 2 and 3—These outputs enable peripheral or memory devices at programmed addresses if they are appropriately defined. The double drive capability for CS2 and CS3 is independently defined for each signal in the SIUMCR.
UPWAITA GPL_A4	Hi-Z	C1	Bidirectional	User Programmable Machine Wait A—This input is sampled as defined by the user when an access to an external slave is controlled by the UPMA. General-Purpose Line 4 on UPMA—This output reflects the value specified in the UPMA when an external transfer to a slave is controlled by UPMA.
UPWAITB GPL_B4	Hi-Z	B1	Bidirectional	User Programmable Machine Wait B—This input is sampled as defined by the user when an access to an external slave is controlled by the UPMB. General-Purpose Line 4 on UPMB—This output reflects the value specified in the UPMB when an external transfer to a slave is controlled by UPMB.
GPL_A5	High	D3	Output	General-Purpose Line 5 on UPMA—This output reflects the value specified in the UPMA when an external transfer to a slave is controlled by UPMA. This signal can also be controlled by the UPMB.
PORESET	Hi-Z	R2	Input	Power on Reset—When asserted, this input causes the MPC855T to enter the power-on reset state.

Table 12-1. Signal Descriptions (continued)

Name	Hard Reset	Number	Type	Description
RSTCONF	Hi-Z	P3	Input	Reset Configuration—The MPC855T samples this input while HRESET is asserted. If RSTCONF is asserted, the configuration mode is sampled in the form of the hard reset configuration word driven on the data bus. When RSTCONF is negated, the MPC855T uses the default configuration mode. Note that the initial base address of internal registers is determined in this sequence.
HRESET	Low	N4	Open-drain	Hard Reset—Asserting this open drain signal puts the MPC855T in hard reset state.
SRESET	Low	P2	Open-drain	Soft Reset—Asserting this open drain line puts the MPC855T in soft reset state.
XTAL	Analog Driving	P1	Analog Output	This output is one of the connections to an external crystal for the internal oscillator circuitry.
EXTAL	Hi-Z	N1	Analog Input (3.3V only)	This line is one of the connections to an external crystal for the internal oscillator circuitry.
XFC	Analog Driving	T2	Analog Input	External Filter Capacitance—This input is the connection pin for an external capacitor filter for the PLL circuitry.
CLKOUT	Note [2]	W3	Output	Clock Out—This output is the clock system frequency.
EXTCLK	Hi-Z	N2	Input (3.3V only)	External Clock—This input is the external input clock from an external source.
TEXP	High	N3	Output	Timer Expired—This output reflects the status of PLPRCR[TEXPS].
ALE_A	Low	K2	Output	Address Latch Enable A—This output line is asserted when MPC855T initiates an access to a region under the control of the PCMCIA interface to socket A.
CE1_A	High	B3	Output	Card Enable 1 Slot A—This output signal enables even byte transfers when accesses to PCMCIA Slot A are handled under the control of the PCMCIA interface.
CE2_A	High	A3	Output	Card Enable 2 Slot A—This output signal enables odd byte transfers when accesses to PCMCIA Slot A are handled under the control of the PCMCIA interface.
WAIT_A	Hi-Z	R3	Input	Wait Slot A—This input signal, if asserted low, causes a delay in the completion of a transaction on the PCMCIA controlled Slot A.
WAIT_B	Hi-Z	R4	Input	Wait Slot B—This input, if asserted low, causes a delay in the completion of a transaction on the PCMCIA controlled Slot B.
IP_A(0)	Hi-Z	T5	Input	Input Port A 0—This input signals is monitored by the MPC855T and its value is reflected in the PIPR and PSCR of the PCMCIA interface.
IP_A(1)	Hi-Z	T4	Input	Input Port A 1—This input signals is monitored by the MPC855T and its value is reflected in the PIPR and PSCR of the PCMCIA interface.

Table 12-1. Signal Descriptions (continued)

Name	Hard Reset	Number	Type	Description
IP_A2 $\overline{\text{IOIS16\_A}}$	Hi-Z	U3	Input	Input Port A 2—This input signal is monitored by the MPC855T and its value and changes are reported in the PIPR and PSCR of the PCMCIA interface. I/O Device A is 16 Bits Ports Size—This input signal is monitored by the MPC855T when a transaction under the control of the PCMCIA interface is initiated to an I/O region in socket A of the PCMCIA space.
IP_A(3)	Hi-Z	W2	Input	Input Port A 3—This input signals is monitored by the MPC855T and its value is reflected in the PIPR and PSCR of the PCMCIA interface.
IP_A(4)	Hi-Z	U4	Input	Input Port A 4—This input signals is monitored by the MPC855T and its value is reflected in the PIPR and PSCR of the PCMCIA interface.
IP_A(5)	Hi-Z	U5	Input	Input Port A 5—This input signals is monitored by the MPC855T and its value is reflected in the PIPR and PSCR of the PCMCIA interface
IP_A(6)	Hi-Z	T6	Input	Input Port A 6—This input signals is monitored by the MPC855T and its value is reflected in the PIPR and PSCR of the PCMCIA interface.
IP_A(7)	Hi-Z	T3	Input	Input Port A 7—This input signals is monitored by the MPC855T and its value is reflected in the PIPR and PSCR of the PCMCIA interface.
ALE_B DSCK/AT1	See Table 12-2	J1	Bidirectional Three-state	Address Latch Enable B—This output is asserted when the MPC855T initiates an access to a region under the control of the PCMCIA socket B interface. Development Serial Clock—This input is the clock for the debug port interface. Address Type 1—The MPC855T drives this bidirectional three-state line when it initiates a transaction on the external bus. When the transaction is initiated by the core, it indicates if the transfer is for user or supervisor state. This signal is not used for transactions initiated by external masters.
IP_B[0–1] IWP[0–1] VFLS[0–1]	See Table 12-2	H2, J3	Bidirectional	Input Port B 0–1—The MPC855T senses these inputs; their values and changes are reported in the PIPR and PSCR of the PCMCIA interface. Instruction Watchpoint 0-1—These outputs report the detection of an instruction watchpoint in the program flow executed by the core. Visible History Buffer Flushes Status—The MPC855T outputs VFLS[0–1] when program instruction flow tracking is required. They report the number of instructions flushed from the history buffer in the core.

Table 12-1. Signal Descriptions (continued)

Name	Hard Reset	Number	Type	Description
IP_B2 IOIS16_B AT2	Hi-Z	J2	Bidirectional Three-state	Input Port B 2—The MPC855T senses this input; its value and changes are reported in the PIPR and PSCR of the PCMCIA interface. I/O Device B is 16 Bits Port Size—The MPC855T monitors this input when a PCMCIA interface transaction is initiated to an I/O region in socket B in the PCMCIA space. Address Type 2—The MPC855T drives this bidirectional three-state signal when it initiates a transaction on the external bus. If the core initiates the transaction, it indicates if the transfer is instruction or data. This signal is not used for transactions initiated by external masters.
IP_B3 IWP2 VF2	See Table 12-2	G1	Bidirectional	Input Port B 3—The MPC855T monitors this input; its value and changes are reported in the PIPR and PSCR of the PCMCIA interface. Instruction Watchpoint 2—This output reports the detection of an instruction watchpoint in the program flow executed by the core. Visible Instruction Queue Flush Status—The MPC855T outputs VF2 with VF0/VF1 when instruction flow tracking is required. VF*n* reports the number of instructions flushed from the instruction queue in the core.
IP_B4 LWP0 VF0	Hi-Z	G2	Bidirectional	Input Port B 4—The MPC855T monitors this input; its value and changes are reported in the PIPR and PSCR of the PCMCIA interface. Load/Store Watchpoint 0—This output reports the detection of a data watchpoint in the program flow executed by the core. Visible Instruction Queue Flushes Status—The MPC855T outputs VF0 with VF1/VF2 when instruction flow tracking is required. VF*n* reports the number of instructions flushed from the instruction queue in the core.
IP_B5 LWP1 VF1	Hi-Z	J4	Bidirectional	Input Port B 5—The MPC855T monitors this input; its value and changes are reported in the PIPR and PSCR of the PCMCIA interface. Load/Store Watchpoint 1—This output reports the detection of a data watchpoint in the program flow executed by the core. Visible Instruction Queue Flushes Status—The MPC855T outputs VF1 with VF0 and VF2 when instruction flow tracking is required. VF*n* reports the number of instructions flushed from the instruction queue in the core.

Table 12-1. Signal Descriptions (continued)

Name	Hard Reset	Number	Type	Description
IP_B6 DSDI AT0	Hi-Z	K3	Bidirectional Three-state	Input Port B 6—The MPC855T senses this input and its value and changes are reported in the PIPR and PSCR of the PCMCIA interface. See Chapter 16, “PCMCIA Interface.” Development Serial Data Input—Data input for the debug port interface. See Chapter 44, “System Development and Debugging.” Address Type 0—The MPC855T drives this bidirectional three-state line 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 CPU. This signal is not used for transactions initiated by external masters.
IP_B7 $\overline{\text{PTR}}$ AT3	Hi-Z	H1	Bidirectional Three-state	Input Port B 7—The MPC855T monitors this input; its value and changes are reported in the PIPR and PSCR of the PCMCIA interface. Program Trace—To allow program flow tracking, the MPC855T asserts this output to indicate an instruction fetch is taking place. Address Type 3—The MPC855T drives the bidirectional three-state signal when it starts a transaction on the external bus. When the core initiates a transfer, AT3 indicates whether it is a reservation for a data transfer or a program trace indication for an instruction fetch. This signal is not used for transactions initiated by external masters.
OP0	Low	L4	Output	Output Port 0—This output signals is generated by the MPC855T as a result of a write to the PGCRA register in the PCMCIA interface.
OP1	Low	L2	Output	Output Port 1—The MPC855T generates these outputs as a result of a write to the PGCRA register in the PCMCIA interface.
OP2 MODCK1 $\overline{\text{STS}}$	Hi-Z	L1	Bidirectional	Output Port 2—This output is generated by the MPC855T as a result of a write to the PGCRB register in the PCMCIA interface. Mode Clock 1—Input sampled when $\overline{\text{PORESET}}$ is negated to configure PLL/clock mode. Special Transfer Start—The MPC855T drives this output to indicate the start of an external bus transfer or of an internal transaction in show-cycle mode.
OP3 MODCK2 DSDO	Hi-Z	M4	Bidirectional	Output Port 3—This output is generated by the MPC855T as a result of a write to the PGCRB register in the PCMCIA interface. Mode Clock 2—This input is sampled at the $\overline{\text{PORESET}}$ negation to configure the PLL/clock mode of operation. Development Serial Data Output—Output data from the debug port interface.

Table 12-1. Signal Descriptions (continued)

Name	Hard Reset	Number	Type	Description
BADDR30 REG	Hi-Z	K4	Output	Burst Address 30—This output duplicates the value of A30 when the following is true: • An internal master in the MPC855T initiates a transaction on the external bus. • An asynchronous external master initiates a transaction. • A synchronous external master initiates a single beat transaction. The memory controller uses BADDR30 to increment the address lines that connect to memory devices when a synchronous external master or an internal master initiates a burst transfer. Register—When an internal master initiates an access to a slave under control of the PCMCIA interface, this signal duplicates the value of TSIZ0/REG. When an external master initiates an access, REG is output by the PCMCIA interface (if it must handle the transfer) to indicate the space in the PCMCIA card being accessed.
BADDR[28–29]	Hi-Z	M3 M2	Output	Burst Address—Outputs that duplicate A[28–29] values when one of the following occurs: • An internal master in the MPC855T initiates a transaction on the external bus. • An asynchronous external master initiates a transaction. • A synchronous external master initiates a single beat transaction. The memory controller uses these signals to increment the address lines that connect to memory devices when a synchronous external or internal master starts a burst transfer.
AS	Hi-Z	L3	Input	Address Strobe—Input driven by an external asynchronous master to indicate a valid address on A[0–31]. The MPC855T memory controller synchronizes AS and controls the memory device addressed under its control.
PA[15] RXD1	Hi-Z	C18	Bidirectional	General-Purpose I/O Port A Bit 15—Bit 15 of the general-purpose I/O port A. RXD1—Receive data input for SCC1.
PA[14] TXD1 TXD4	Hi-Z	D17	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port A Bit 14—Bit 14 of the general-purpose I/O port A. TXD1—Transmit data output for SCC1. TXD4—Transmit data output for SCC4.
PA[13]	Hi-Z	E17	Bidirectional	General-Purpose I/O Port A Bit 13—Bit 13 of the general-purpose I/O port A.
PA[12]	Hi-Z	F17	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port A Bit 12—Bit 12 of the general-purpose I/O port A.
PA[11]	Hi-Z	G16	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port A Bit 11—Bit 11 of the general-purpose I/O port A.

Table 12-1. Signal Descriptions (continued)

Name	Hard Reset	Number	Type	Description
PA[10] TXD3	Hi-Z	J17	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port A Bit 10—Bit 10 of the general-purpose I/O port A. TXD3—Transmit data output for SCC3.
PA[9] L1TXDA RXD4	Hi-Z	K18	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port A Bit 11—Bit 9 of the general-purpose I/O port A. L1TXDA—Transmit data output for the serial interface TDMa. RXD4—Receive data input for SCC4.
PA[8] L1RXDA TXD4	Hi-Z	L17	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port A Bit 8—Bit 8 of the general-purpose I/O port A. L1RXDA—Receive data input for the serial interface TDMa. TXD4—Transmit data output for SCC4.
PA[7] CLK1 TIN1 L1RCLKA BRGO1	Hi-Z	M19	Bidirectional	General-Purpose I/O Port A Bit 7—Bit 7 of the general-purpose I/O port A. CLK1—One of eight clock inputs that can be used to clock SCCs and SMCs. TIN1—Timer 1 external clock. L1RCLKA—Receive clock for the serial interface TDMa. BRGO1—Output clock of BRG1.
PA[6] CLK2 $\overline{\text{TOUT1}}$	Hi-Z	M17	Bidirectional	General-Purpose I/O Port A Bit 6—Bit 6 of the general-purpose I/O port A. CLK2—One of eight clock inputs that can be used to clock SCCs and SMCs. CLK2 can also be used as a clock source for the BRGs. $\overline{\text{TOUT1}}$—Timer 1 output.
PA[5] CLK3 TIN2 L1TCLKA BRGO2	Hi-Z	N18	Bidirectional	General-Purpose I/O Port A Bit 5—Bit 5 of the general-purpose I/O port A. CLK3—One of eight clock inputs that can be used to clock SCCs and SMCs. TIN2—Timer 2 external clock input. L1TCLKA—Transmit clock for the serial interface TDMa. BRGO2—Output clock of BRG2.
PA[4] CLK4 $\overline{\text{TOUT2}}$	Hi-Z	P19	Bidirectional	General-Purpose I/O Port A Bit 4—Bit 4 of the general-purpose I/O port A. CLK4—One of eight clock inputs that can be used to clock SCCs and SMCs. $\overline{\text{TOUT2}}$—Timer 2 output.
PA[3] CLK5 TIN3 BRGO3	Hi-Z	P17	Bidirectional	General-Purpose I/O Port A Bit 3—Bit 3 of the general-purpose I/O port A. CLK5—One of eight clock inputs that can be used to clock SCCs and SMCs. TIN3—Timer 3 external clock input. BRGO3—Output clock of BRG3.
PA[2] CLK6 $\overline{\text{TOUT3}}$	Hi-Z	R18	Bidirectional	General-Purpose I/O Port A Bit 2—Bit 2 of the general-purpose I/O port A. CLK6—One of eight clock inputs that can be used to clock the SCCs and SMCs. CLK6 can also be used as a clock source for the BRGs. $\overline{\text{TOUT3}}$—Timer 3 output.

Table 12-1. Signal Descriptions (continued)

Name	Hard Reset	Number	Type	Description
PA[1] CLK7 TIN4 BRGO4	Hi-Z	T19	Bidirectional	General-Purpose I/O Port A Bit 1—Bit 1 of the general-purpose I/O port A. CLK7—One of eight clock inputs that can be used to clock SCCs and SMCs. TIN4—Timer 4 external clock input. BRGO4—BRG4 output clock.
PA[0] CLK8 $\overline{\text{TOUT4}}$	Hi-Z	U19	Bidirectional	General-Purpose I/O Port A Bit 0—Bit 0 of the general-purpose I/O port A. CLK8—One of eight clock inputs that can be used to clock SCCs and SMCs. $\overline{\text{TOUT4}}$—Timer 4 output.
PB[31] $\overline{\text{SPISEL}}$ $\overline{\text{REJECT1}}$	Hi-Z	C17	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port B Bit 31—Bit 31 of the general-purpose I/O port B. $\overline{\text{SPISEL}}$—SPI slave select input. $\overline{\text{REJECT1}}$—SCC1 CAM interface reject pin.
PB[30] SPICLK $\overline{\text{RSTRT2}}$	Hi-Z	C19	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port B Bit 30—Bit 30 of the general-purpose I/O port B. SPICLK—SPI output clock when it is configured as a master or SPI input clock when it is configured as a slave. $\overline{\text{RSTRT2}}$—SCC2 serial CAM interface output signal that marks the start of a frame.
PB[29] SPIMOSI	Hi-Z	E16	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port B Bit 29—Bit 29 of the general-purpose I/O port B. SPIMOSI—SPI output data when it is configured as a master or SPI input data when it is configured as a slave.
PB[28] SPIMISO BRGO4	Hi-Z	D19	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port B Bit 28—Bit 29 of the general-purpose I/O port B. SPIMISO—SPI input data when the MPC855T is a master; SPI output data when it is a slave. BRGO4—BRG4 output clock.
PB[27] I2CSDA BRGO1	Hi-Z	E19	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port B Bit 27—Bit 27 of the general-purpose I/O port B. I2CSDA—I^2C serial data pin. Bidirectional; should be configured as an open-drain output. BRGO1—BRG1 output clock.
PB[26] I2CSCL BRGO2	Hi-Z	F19	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port B Bit 26—Bit 26 of the general-purpose I/O port B. I2CSCL—I^2C serial clock pin. Bidirectional; should be configured as an open-drain output. BRGO2—BRG2 output clock.
PB[25] SMTXD1	Hi-Z	J16	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port B Bit 25—Bit 25 of the general-purpose I/O port B. SMTXD1—SMC1 transmit data output.
PB[24] SMRXD1	Hi-Z	J18	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port B Bit 24—Bit 24 of the general-purpose I/O port B. SMRXD1—SMC1 receive data input.

Table 12-1. Signal Descriptions (continued)

Name	Hard Reset	Number	Type	Description
PB[23] SMSYN1 SDACK1	Hi-Z	K17	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port B Bit 23—Bit 23 of the general-purpose I/O port B. SMSYN1—SMC1 external sync input. SDACK1—SDMA acknowledge 1 output that is used as a peripheral interface signal for IDMA emulation, or as a CAM interface signal for Ethernet.
PB[22] SMSYN2 SDACK2	Hi-Z	L19	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port B Bit 22—Bit 22 of the general-purpose I/O port B. SMSYN2—SMC2 external sync input. SDACK2—SDMA acknowledge 2 output that is used as a peripheral interface signal for IDMA emulation, or as a CAM interface signal for Ethernet.
PB[21] SMTXD2 PHSEL[1]	Hi-Z	K16	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port B Bit 21—Bit 21 of the general-purpose I/O port B. SMTXD2—SMC2 transmit data output. PHSEL[1]—Least significant bit of PHY select bus.
PB[20] SMRXD2 L1CLKOA PHSEL[0]	Hi-Z	L16	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port B Bit 20—Bit 20 of the general-purpose I/O port B. SMRXD2—SMC2 receive data input. L1CLKOA—Clock output from the serial interface TDMa. PHSEL[0]—Most significant bit of PHY select bus.
PB[19] RTS1 L1ST1	Hi-Z	N19	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port B Bit 19—Bit 19 of the general-purpose I/O port B. RTS1—Request to send modem line for SCC1. L1ST1—One of four output strobes that can be generated by the serial interface.
PB[18] L1ST2	Hi-Z	N17	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port B Bit 18—Bit 18 of the general-purpose I/O port B. L1ST2—One of four output strobes that can be generated by the serial interface.
PB[17] L1ST3 PHREQ[1]	Hi-Z	P18	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port B Bit 17—Bit 17 of the general-purpose I/O port B. L1ST3—One of four output strobes that can be generated by the serial interface. PHREQ[1]—Least significant bit of PHY request bus.
PB[16] L1RQa L1ST4 PHREQ[0]	Hi-Z	N16	Bidirectional (Optional: Open-drain)	General-Purpose I/O Port B Bit 16—Bit 16 of the general-purpose I/O port B. L1RQa—D-channel request signal for serial interface TDMa. L1ST4—One of four output strobes that can be generated by the serial interface. PHREQ[0]—Most significant bit of PHY request bus.
PB[15] BRGO3 Txclav	Hi-Z	R17	Bidirectional	General-Purpose I/O Port B Bit 15—Bit 15 of the general-purpose I/O port B. BRGO3—BRG3 output clock. Txclav—Transmit cell available input signal.

Table 12-1. Signal Descriptions (continued)

Name	Hard Reset	Number	Type	Description
PB[14] RSTRT1	Hi-Z	U18	Bidirectional	General-Purpose I/O Port B Bit 14—Bit 14 of the general-purpose I/O port B. RSTRT1—SCC1 serial CAM interface outputs that marks the start of a frame.
PC[15] DREQ0 RTS1 L1ST1 Rxclav	Hi-Z	D16	Bidirectional	General-Purpose I/O Port C Bit 15—Bit 15 of the general-purpose I/O port C. DREQ0—IDMA channel 1 request input. RTS1—Request to send modem line for SCC1. L1ST1—One of four output strobes that can be generated by the serial interface. Rxclav—Receive cell available input signal
PC[14] DREQ1 L1ST2	Hi-Z	D18	Bidirectional	General-Purpose I/O Port C Bit 14—Bit 14 of the general-purpose I/O port C. DREQ1—IDMA channel 2 request input. L1ST2—One of four output strobes that can be generated by the serial interface.
PC[13] L1ST3	Hi-Z	E18	Bidirectional	General-Purpose I/O Port C Bit 13—Bit 13 of the general-purpose I/O port C. L1ST3—One of four output strobes that can be generated by the serial interface.
PC[12] L1RQa L1ST4	Hi-Z	F18	Bidirectional	General-Purpose I/O Port C Bit 12—Bit 12 of the general-purpose I/O port C. L1RQa—D-channel request signal for serial interface TDMa. L1ST4—One of four output strobes that can be generated by the serial interface.
PC[11] CTS1	Hi-Z	J19	Bidirectional	General-Purpose I/O Port C Bit 11—Bit 11 of the general-purpose I/O port C. CTS1—Clear to send modem line for SCC1.
PC[10] CD1 TGATE1	Hi-Z	K19	Bidirectional	General-Purpose I/O Port C Bit 10—Bit 10 of the general-purpose I/O port C. CD1—Carrier detect modem line for SCC1. TGATE1—Timer 1/timer 2 gate signal.
PC[9]	Hi-Z	L18	Bidirectional	General-Purpose I/O Port C Bit 9—Bit 9 of the general-purpose I/O port C.
PC[8] TGATE2	Hi-Z	M18	Bidirectional	General-Purpose I/O Port C Bit 8—Bit 8 of the general-purpose I/O port C. TGATE2—Timer 3/timer 4 gate signal.
PC[7] SDACK2	Hi-Z	M16	Bidirectional	General-Purpose I/O Port C Bit 7—Bit 7 of the general-purpose I/O port C. SDACK2—SDMA acknowledge 2 output that is used as a peripheral interface signal for IDMA emulation or as a CAM interface signal for Ethernet.
PC[6]	Hi-Z	R19	Bidirectional	General-Purpose I/O Port C Bit 6—Bit 6 of the general-purpose I/O port C.

Table 12-1. Signal Descriptions (continued)

Name	Hard Reset	Number	Type	Description
PC[5] L1TSYNCA SDACK1	Hi-Z	T18	Bidirectional	General-Purpose I/O Port C Bit 5—Bit 5 of the general-purpose I/O port C. L1TSYNCA—Transmit sync input for serial interface TDMa. SDACK1—SDMA acknowledge 1output that is used as a peripheral interface signal for IDMA emulation or as a CAM interface signal for Ethernet.
PC[4] L1RSYNCA	Hi-Z	T17	Bidirectional	General-Purpose I/O Port C Bit 4—Bit 4 of the general-purpose I/O port C. L1RSYNCA—Receive sync input for serial interface TDMa.
PD[15] L1TSYNCA UTPB[0] MII-RXD3	Hi-Z	U17	Bidirectional	General-Purpose I/O Port D Bit 15—Bit 15 of the general-purpose I/O port D. L1TSYNCA—Input transmit data sync signal to the TDM channel A. MII-RXD3—Message independent interface receive data 3. UTPB[0]—UTOPIA bus bit 0 input/output signal.
PD[14] L1RSYNCA UTPB[1] MII-RXD2	Hi-Z	V19	Bidirectional	General-Purpose I/O Port D Bit 14—Bit 14 of the general-purpose I/O port D. L1RSYNCA—Input receive data sync signal to the TDM channel A. MII-RXD2—Message independent interface receive data 2. UTPB[1]—UTOPIA bus bit 1 input/output signal.
PD[13] L1TSYNCB UTPB[2] MII-RXD1	Hi-Z	V18	Bidirectional	General-Purpose I/O Port D Bit 13—Bit 13 of the general-purpose I/O port D. L1TSYNCB—Input transmit data sync signal to the TDM channel B. MII-RXD1—Message independent interface receive data 1. UTPB[2]—UTOPIA bus bit 2 input/output signal.
PD[12] UTPB[3] MII-MDC	Hi-Z	R16	Bidirectional	General-Purpose I/O Port D Bit 12—Bit 12 of the general-purpose I/O port D.. MII-MDC—Message independent interface management data clock. UTPB[3]—UTOPIA bus bit 3 input/output signal.
PD[11] RXENB MII-TX-ERR	Hi-Z	T16	Bidirectional	General-Purpose I/O Port D Bit 11—Bit 11 of the general-purpose I/O port D. RXENB—Receive enable output signal. MII-TX-ERR—Media independent interface transmit error
PD[10] TXENB MII-RXD0	Hi-Z	W18	Bidirectional	General-Purpose I/O Port D Bit 10—Bit 10 of the general-purpose I/O port D. TXENB—Transmit enable output signal. MII-RXD0—Media independent interface receive data 0

Table 12-1. Signal Descriptions (continued)

Name	Hard Reset	Number	Type	Description
PD[9] UTPCLK MII-TXD0	Hi-Z	V17	Bidirectional	General-Purpose I/O Port D Bit 9—Bit 9 of the general-purpose I/O port D. UTPCLK—UTOPIA Clock input/output signal. The direction of this I/O pin in non muxed UTOPIA mode is defined by UTOPIA mode register. As an input or output the frequency of the UTOPIA clock can be up to 50Mhz and in the following range: SYSCLK > UTPCLK > SYSCLK/10 MII-TXD0—Media independent interface transmit data 0
PD[8] MII-RXCLK	Hi-Z	W17	Bidirectional	General-Purpose I/O Port D Bit 8—Bit 8 of the general-purpose I/O port D. MII-RXCLK—Media independent interface receive clock
PD[7] UTPB[4] MII-RX-ERR	Hi-Z	T15	Bidirectional	General-Purpose I/O Port D Bit 7—Bit 7 of the general-purpose I/O port D. UTPB[4]—UTOPIA bus bit 4 input/output signal. MII-RX-ERR—Message independent interface receive error.
PD[6] UTPB[5] MII-RXDV	Hi-Z	V16	Bidirectional	General-Purpose I/O Port D Bit 6—Bit 6 of the general-purpose I/O port D. UTPB[5]—UTOPIA bus bit 5 input/output signal. MII-RXDV—Message independent interface receive data valid.
PD[5] $\overline{\text{REJECT2}}$ UTPB[6] MII-TXD3	Hi-Z	U15	Bidirectional	General-Purpose I/O Port D Bit 5—Bit 5 of the general-purpose I/O port D. $\overline{\text{REJECT2}}$—This input to SCC2 allows a CAM to reject the current Ethernet frame after it determines the frame address did not match. UTPB[6]—UTOPIA bus bit 6 input/output signal. MII-TXD3—Message independent interface transmit data 3.
PD[4] $\overline{\text{REJECT3}}$ UTPB[7] MII-TXD2	Hi-Z	U16	Bidirectional	General-Purpose I/O Port D Bit 4—Bit 4 of the general-purpose I/O port D. $\overline{\text{REJECT3}}$—This input to SCC3 allows a CAM to reject the current Ethernet frame after it determines the frame address did not match. UTPB[7]—UTOPIA bus bit 7 input/output signal. (most significant bit of UTPB) MII-TXD2—Message independent interface transmit data 2.
PD[3] $\overline{\text{REJECT4}}$ SOC MII-TXD1	Hi-Z	W16	Bidirectional	General-purpose I/O Port D Bit 3—Bit 3 of the general-purpose I/O port D. $\overline{\text{REJECT4}}$—This input to SCC4 allows a CAM to reject the current Ethernet frame after it determines the frame address did not match. SOC—Start of cell input/output signal. MII-TXD1—Message independent interface transmit data 1.
TCK DSCK	Hi-Z	H16	Input	Provides clock to scan chain logic or for the development port logic.

Table 12-1. Signal Descriptions (continued)

Name	Hard Reset	Number	Type	Description
TMS	Pulled up	G18	Input	Controls the scan chain test mode operations.
TDI DSDI	Pulled up	H17	Input	Input serial data for either the scan chain logic or the development port and determines the operating mode of the development port at reset.
TDO DSDO	Low	G17	Output	Output serial data for either the scan chain logic or for the development port.
$\overline{\text{TRST}}$	Pulled up[3]	G19	Input	Test reset for the JTAG scan chain logic.
MII_CRS	Hi-Z	B7	Input	MII carrier receive sense
MII_MDIO	Hi-Z	H18	Bidirectional	MII management data
MII_TXEN	Low	V15	Output	MII transmit enable
MII_COL	Hi-Z	H4	Input	MII collision
Power Supply		See Figure	Power	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 PLL circuitry. KAPWR—Power supply of the internal OSCM, RTC, PIT, DEC, and TB. VSS—Ground for circuits, except for the PLL circuitry. VSSSYN, VSSSYN1—Ground for the PLL circuitry.

1 Pulled Low if $\overline{\text{RSTCONF}}$ pulled down

2 High until SPLL locked, then oscillating

3 See Section 11.4, “TRST and Power Mode Considerations,” and Section 45.6, “Recommended TAP Configuration.”

NOTE:

The reset behavior of a subset of multiple-function pins depends on which signal function is active. The SIUMCR programming determines which signal functions of this pin subset are activated at reset; see Section 10.4.2, “SIU Module Configuration Register (SIUMCR).” Some (but not all) of the SIUMCR default values are determined by the user-controlled hardware reset configuration word; see Section 11.3.1.1, “Hard Reset Configuration Word.” When $\overline{\text{HRESET}}$ (or $\overline{\text{PORESET}}$) is asserted, these pins immediately begin functioning as the signals selected in the SIUMCR. The behavior of these signals is shown in Table 12-2.

Table 12-2. Configuration-Dependent Signal Behavior during Reset

Signal Function Determined at Reset by…		Pin	Signal Behavior
SRESET	**HRESET (or PORESET)**		
Previous programming of SIUMCR	SIUMCR default values only	BDIP/GPL_B5	BDIP: high impedance GPL_B5: high[1]
		RSV/IRQ2	IRQ2: high impedance RSV: high[1]
		CR/IRQ3	IRQ3: high impedance CR: high[1]
		KR/RETRY/IRQ4/SPKROUT	IRQ4: high impedance KR/RETRY: high impedance[1] SPKROUT: low[1]
		FRZ/IRQ6	FRZ: low IRQ6: high impedance[1]
	SIUMCR default values as driven by the hard reset configuration word	ALE_B/DSCK/AT1	ALE_B: low DSCK/AT1: high impedance
		IP_B[0–1]/IWP[0–1]/VFLS[0–1]	IP_B[0–1]: high impedance. IWP[0–1]: high VFLS[0–1]: low
		IP_B3/IWP2/VF2	IP_B3: high impedance IWP2: high VF2: low
		IP_B4/LWP0/VF0	IP_B4: high impedance LWP0: high VF0: low
		IP_B5/LWP1/VF1	IP_B5: high impedance LWP1: high; VF1: low

1 After a hard reset, this signal function is actually inactive until the user selects the function by programming the SIUMCR.

12.2 Active Pull-Up Buffers

Active pull-up buffers are a special variety of bidirectional three-state buffer with the following properties:

- When enabled as an output and driving low, they behave as a normal output driver (that is, the pin is constantly driven low).
- When enabled as an output and driving high, drive high until an internal detection circuit determines that the output has reached the logic high threshold and then stop driving (that is, the pin switches to high-impedance).
- When disabled as an output or functioning as an input, it is not driven.

Due to the behavior of the buffer when being driven high, a pull-up resistor is required externally to function as a ‘bus keep’ for these shared signals in periods when no drivers are active and to keep the buffer from oscillating when the buffer is driving high, because

if the voltage ever dips below the logic high threshold while the buffer is enabled as an output, the buffer will reactivate. Further, external logic must not attempt to drive these signals low while active pull-up buffers are enabled as outputs, because the buffers will reactivate and drive high, resulting in a buffer fight and possible damage to the MPC855T, to the system, or to both.

Figure 12-4 compares three-state buffers and active pull-up buffers graphically in general terms. It makes no implication as to which edges trigger which events for any particular signal.

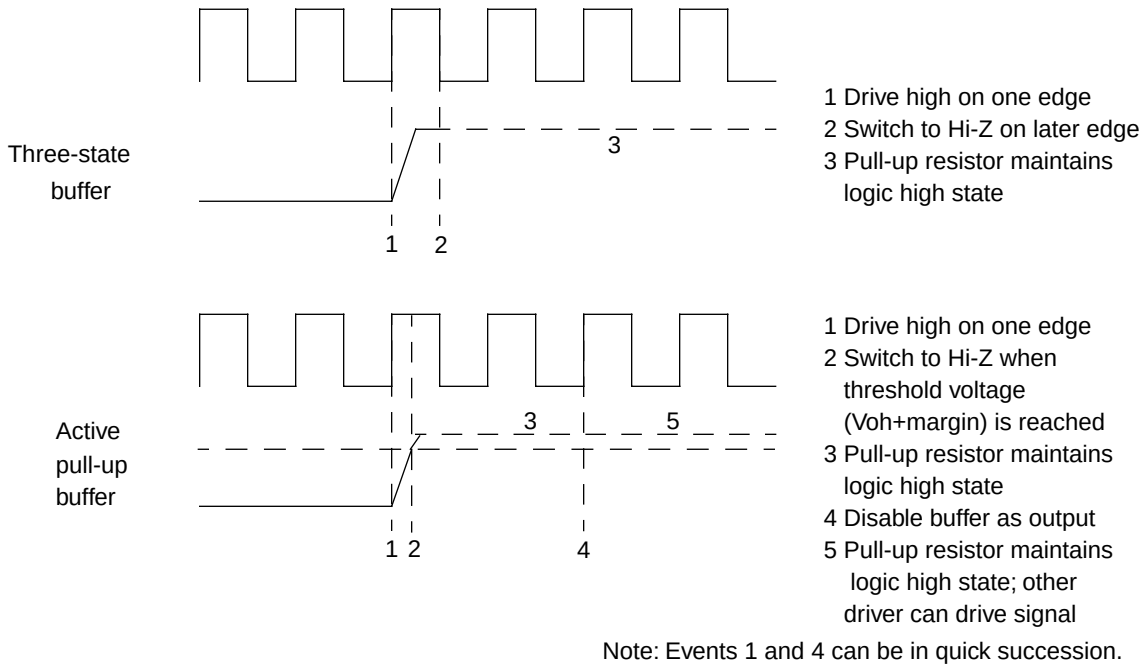


Figure 12-4. Three-State Buffers and Active Pull-Up Buffers

Table 12-3 summarizes when active pull-up drivers are enabled as outputs.

Table 12-3. Active Pull-Up Resistors Enabled as Outputs

Signal	Description
$\overline{TS}$, $\overline{BB}$	When the MPC855T is the external bus master throughout the entire bus cycle.
$\overline{BI}$	When the MPC855T's memory controller responds to the access on the external bus, throughout the entire bus cycle.
$\overline{TA}$	When the MPC855T's memory controller responds to the access on the external bus, then: • •For chip selects controlled by a GPCM set for external $\overline{TA}$, the MPC860's $\overline{TA}$ buffer is not enabled as an output. • •For chip-selects controlled by the GPCM set to terminate in n wait-states, $\overline{TA}$ is enabled as an output on cycle (n-1) and driven high, then is driven low on cycle n, terminating the bus transaction. External logic can drive $\overline{TA}$ at any point before this, thus terminating the cycle early. [For example, assume the GPCM is programmed to drive $\overline{TA}$ after 15 cycles. If external logic drives $\overline{TA}$ before 14 clocks have elapsed then the $\overline{TA}$ is accepted by the MPC860 as a cycle termination.] • •For UPM-controlled chip selects, the $\overline{TA}$ buffer is enabled as an output throughout the entire bus cycle.

The purpose of active pull-up buffers is to allow access to zero wait-state logic that drives a shared signal on the clock cycle immediately following a cycle in which the signal is driven by the MPC855T. In other words, it eliminates the need for a bus turn-around cycle.

12.3 Internal Pull-Up and Pull-Down Resistors

The TMS and $\overline{\text{TRST}}$ pins have internal pull-up resistors. MPC855T devices from Rev 0 to Rev A.3 (masks xE64C and xF84C) have an internal pull-up resistor on TCK/DSCK but no internal pull-up resistor on TDI/DSDI. This was corrected on Rev B and later; on these chips, the internal pull-up resistor was removed from TCK/DSCK and an internal pull-up resistor was added to TDI/DSDI.

If $\overline{\text{RSTCONF}}$ is pulled down, during hardware reset (initiated by $\overline{\text{HRESET}}$ or $\overline{\text{PORESET}}$), the data bus D[0–31] is pulled down with internal pull-down resistors. These internal pull-down resistors are to provide a logic-zero default for these pins when programming the hard reset configuration word (See Section 11.3.1.1, “Hard Reset Configuration Word.”). These internal pull-down resistors are disconnected after $\overline{\text{HRESET}}$ is negated.

No other pins have internal pull-ups or pull-downs.

Resistance values for internal pull-up and pull-down resistors are not specified because their values may vary due to process variations and shrinks in die size, and they are not tested. Typical values are on the order of 5 kΩ but can vary by approximately a factor of 2.

12.4 Recommended Basic Pin Connections

The following sections provided recommended pin connections.

12.4.1 Reset Configuration

Some external pin configuration is determined at reset by the hard reset configuration word. Thus, some decisions as to system configuration (for example, location of BDM pins) should be made before required application of pull-up and pull-down resistors can be determined.

$\overline{\text{RSTCONF}}$ should be grounded if the hard reset configuration word is used to configure the MPC855T or should be connected to VCC if the default configuration is used.

Pull-up resistors may not be used on D[0–31] to set the hard reset configuration word, as the values of the internal pull-down resistors are not specified or guaranteed. To change a data bus signal from its default logic low state during reset, actively drive that signal high.

MODCK[1–2] must be used to determine the default clocking mode for the MPC855T. After power-on reset, the MODCK[1–2] pins change function and become outputs. Thus, if these alternate functions are also desired, then the MODCK[1–2] configuration should be set with three-state drivers that turn off after $\overline{\text{PORESET}}$ is negated; however, if

MODCK[1–2] pins' alternate output functions are not used in the system, they can be configured with pull-up and pull-down resistors.

12.4.1.1 Bus Control Signals and Interrupts

Signals with open-drain buffers and active pull-up buffers ($\overline{\text{HRESET}}$, $\overline{\text{SRESET}}$, $\overline{\text{TEA}}$, $\overline{\text{TS}}$, $\overline{\text{TA}}$, $\overline{\text{BI}}$, and $\overline{\text{BB}}$) must have external pull-up resistors.

Some other input signals do not absolutely require a pull-up resistor, as they may be actively driven by external logic. However, if they are not used externally, or if the external logic connected to them is not always actively driving, they may need external pull-up resistors to hold them negated. These signals include the following:

- $\overline{\text{PORESET}}$
- $\overline{\text{AS}}$
- $\overline{\text{CR}}$/$\overline{\text{IRQ3}}$
- $\overline{\text{KR}}$/$\overline{\text{RETRY}}$/$\overline{\text{IRQ4}}$/SPKROUT (if configured as $\overline{\text{KR}}$/$\overline{\text{RETRY}}$ or $\overline{\text{IRQ4}}$)
- Any $\overline{\text{IRQx}}$ (if configured as $\overline{\text{IRQx}}$)
- $\overline{\text{BR}}$ (if the MPC860's internal bus arbiter is used)
- $\overline{\text{BG}}$ (if an external bus arbiter is used)

12.4.2 JTAG and Debug Ports

Recommendations on configuration of the JTAG pins (including TMS, $\overline{\text{TRST}}$, TDI, TDO, and TCK) are made in Section 45.6, "Recommended TAP Configuration." See also Section 11.4, "TRST and Power Mode Considerations."

TCK/DSCK or ALE_B/DSCK/AT1 (depending on the configuration of the DSCK function) should be connected to ground through a pull-down resistor to disable debug mode as a default. When required, an external debug-mode controller can actively drive this signal high to put the MPC860 into debug mode.

The two signals TCK/DSCK and TDI/DSDI have special requirements to keep them from oscillating when unused (see Section 12.3, "Internal Pull-Up and Pull-Down Resistors"). Table 12-4 shows the external connection requirements depending on the silicon revision of the MPC860.

Table 12-4. TCK/DSCK and TDI/DSDI Connection Based on MPC860 Revision

Signal	Revision A.3 and Earlier	Revision B and Later
TCK/ DSCK	Should be pulled down to ground; the pull-down resistor must be strong (for example, 1 kΩ) to overcome the internal pull-up resistor.	Should be pulled down to ground. A strong pull-down resistor (for example, 1 kΩ) is recommended.
TDI/ DSDI	Should be pulled up to VCC.	Internal pull-up is provided (no external resistor required).

To allow the application of any version of MPC860, perform the revision A requirements.

12.4.3 Unused Inputs

In general, pull-up resistors should be used on any unused inputs to keep them from oscillating. For example, if PCMCIA is not used, the PCMCIA input pins (WAIT_A, WAIT_B, IP_A[0–8], IP_B[0–8]) should have external pull-up resistors. However, unused pins of port A, B, C, or D can be configured as outputs, and, if they are configured as outputs they do not require external terminations.

12.4.4 Unused Outputs

Unused outputs can be left unterminated.

12.5 Signal States during Reset

During a reset, the signals of the MPC855T behave as shown in Table 12-5.

Table 12-5. General Signal Behavior during Reset

Reset Signal	Signal Behavior
$\overline{\text{HRESET}}$ or $\overline{\text{PORESET}}$	Bus signals are high-impedance.
	Port I/O signals are configured as inputs and are therefore high-impedance.
	Memory controller signals are driven to their inactive state. Refresh stops.
	(For the behavior of specific signals during a hard reset, see Section 12.1, “System Bus Signals.”)
$\overline{\text{SRESET}}$	The current bus cycle aborts. Bus signals revert to their inactive state. (For example, $\overline{\text{BR}}$ or $\overline{\text{BG}}$ negate, and address and data signals become high-impedance.)
	Memory controller aborts the current access, and signals drive to their inactive state (high). Refresh continues.
	Port I/O signals are not re-configured (maintain previous programming).
	SIU pin configuration maintains previous programming; see Table 12-2.

Chapter 13
External Bus Interface

The MPC855T bus is a synchronous, burstable bus that can support multiple masters. Signals driven on this bus are required to make the setup and hold time relative to the bus clock's rising edge. The MPC855T 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 size outputs (TSIZ0, TSIZ1). Access to 16- and 8-bit ports is done for slaves controlled by the memory controller.

13.1 Features

The MPC855T bus interface features are listed as follows:

- 32-bit address bus with transfer size indication
- 32-bit data bus
- Dynamic bus sizing to 32-, 16-, or 8-bit ports accessed through the memory controller
- TTL-compatible interface
- Bus arbitration supported optionally by internal or external logic
- Bus arbitration logic on-chip supports an external master with programmable priority
- Compatible with PowerPC architecture
- Easy to interface to slave devices
- Bus is synchronous (all signals are referenced to rising edge of bus clock)
- Contains support for data parity

13.2 Bus Transfer Overview

The bus transfers information between the MPC855T and external memory or a peripheral device. External devices can accept or provide 8, 16, and 32 bits in parallel and must follow the handshake protocol described in this section. The maximum number of bits accepted or provided during a bus transfer is defined as port width.

The MPC855T's address bus specifies the address for the transfer and its data bus transfers the data. Control signals indicate the beginning of the cycle and the type of cycle, as well as the address space and size of the transfer. The selected device controls cycle length with signal(s) used to terminate the cycle. A strobe signal for the address bus indicates the validity of the address and gives data timing information. The MPC855T bus is synchronous, therefore, the bus and control input signals must be timed to setup and hold times relative to the rising edge of the clock. At minimum, single-beat bus cycles can be completed in two clock cycles.

Furthermore, for all inputs, the MPC855T latches the input's level during a sample window, shown in Figure 13-1, around the rising clock edge. To ensure that an input signal is recognized on a specific rising clock edge, that input must be stable during the sample window. If an input changes during the window, the level recognized by the MPC855T is unpredictable; however, the MPC855T always resolves the latched level to either a logical high or low before using it. For deterministic operation, all input signals must obey the protocols described in this chapter in addition to meeting input setup and hold times.

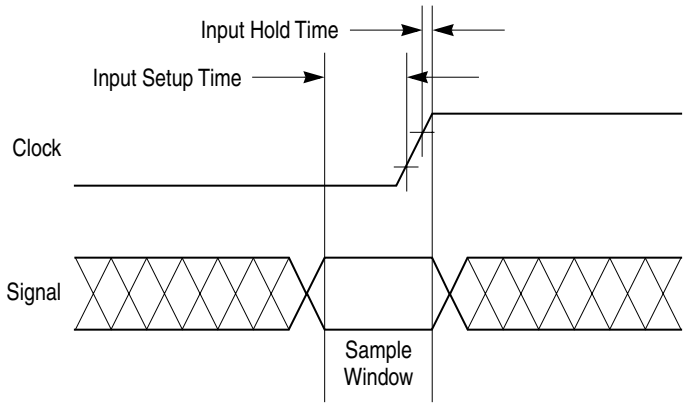


Figure 13-1. Input Sample Window

TSIZ0 and TSIZ1 indicate the number of bytes remaining to be transferred during an operand cycle (consisting of one or more bus cycles) and are driven with the address type signals at the beginning of a bus cycle. These signals are valid at the rising edge of the clock in which the transfer start signal ($\overline{TS}$) is asserted.

13.3 Bus Interface Signal Descriptions

Figure 13-3 shows the bus signals for the MPC855T.

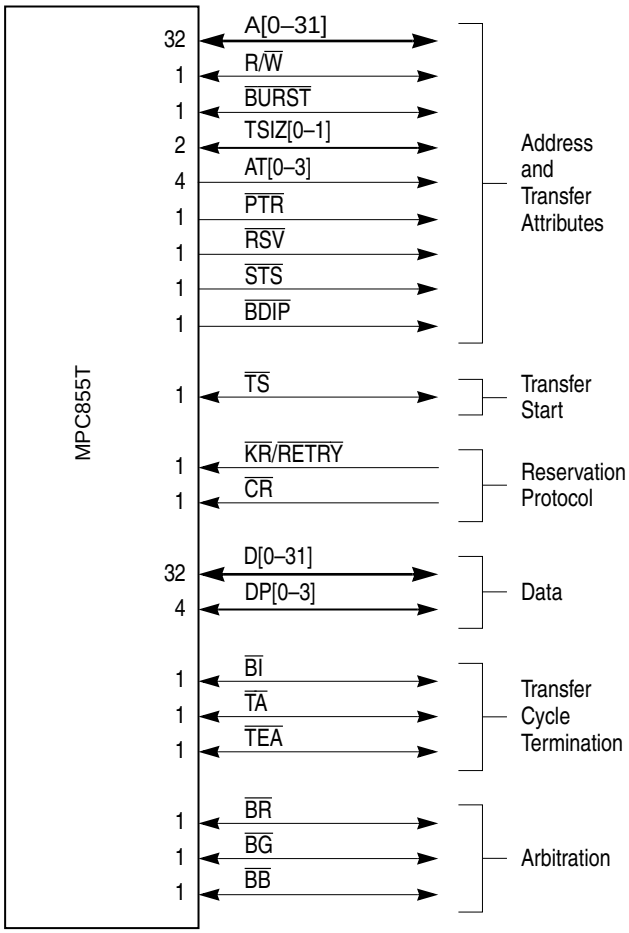


Figure 13-2. MPC855T Bus Signals

Table 13-1 describes each signal; detailed descriptions can be found in subsequent sections.

Table 13-1. MPC855T Signal Overview

Signal	Pins	I/O [1]	Description
Address and Transfer Attributes			
A[–31] Address Bus		O	Driven by the MPC855T when it owns the external bus. Specifies the physical address of the bus transaction. Can change during a transaction when controlled by the memory controller.
		I	Sampled by the MPC855T when an external device initiates a transaction and the memory controller was configured to handle external master accesses.
RD/$\overline{WR}$ Read/Write	1	O	Driven by the MPC855T along with the address when it owns the external bus. Driven high indicates that a read access is in progress. Driven low indicates that a write access is in progress.
		I	Sampled by the MPC855T when an external device initiates a transaction and the memory controller was configured to handle external master accesses.

Table 13-1. MPC855T Signal Overview (continued)

Signal	Pins	I/O [1]	Description
$\overline{\text{BURST}}$ Burst Transfer	1	O	Driven by the MPC855T along with the address when it owns the external bus. Driven low indicates that a burst transfer is in progress. Driven high indicates that the current transfer is not a burst.
		I	Sampled by the MPC855T when an external device initiates a transaction and the memory controller was configured to handle external master accesses.
TSIZ[0–1] Transfer Size	2	O	Driven by the MPC855T along with the address when it owns the external bus. Specifies the data transfer size for the transaction.
		I	Sampled by the MPC855T when an external device initiates a transaction and the memory controller was configured to handle external master accesses.
AT[0–3] Address Type	4	O	Driven by the MPC855T along with the address when it owns the external bus. Indicates additional information about the address on the current transaction.
$\overline{\text{RSV}}$ Reservation Transfer	1	O	Driven by the MPC855T along with the address when it owns the external bus. Indicates additional information about the address on the current transaction.
$\overline{\text{PTR}}$ Program Trace	1	O	Driven by the MPC855T along with the address when it owns the external bus. Indicates additional information about the address on the current transaction.
$\overline{\text{BDIP}}$ Burst Data in Progress	1	O	Driven by the MPC855T when it owns the external bus as part of the burst protocol. Asserted indicates that the second beat in front of the current one is requested by the master. Negated before the burst transfer ends to abort the burst data phase.
Transfer Start			
$\overline{\text{TS}}$ Transfer Start	1	O	Driven by the MPC855T when it owns the external bus. Indicates the start of a transaction on the external bus.
		I	Sampled by the MPC855T when an external device initiates a transaction and the memory controller was configured to handle external master accesses.
$\overline{\text{STS}}$ Special Transfer Start	1	O	Driven by the MPC855T when it owns the external bus. Indicates the start of a transaction on the external bus or signals the beginning of an internal transaction in show cycle mode.
Reservation Protocol			
$\overline{\text{KR}}$/$\overline{\text{RETRY}}$ Kill Reservation/Retry	1	I	If the core initiates a bus cycle by executing a **stwcx**. to a nonlocal bus on which the memory reservation is lost, the nonlocal bus uses this signal to back-off the cycle. See Section 13.4.9, “Memory Reservation.” For regular transactions, the slave device drives this signal to indicate that the MPC855T must relinquish the bus and retry the cycle.

Table 13-1. MPC855T Signal Overview (continued)

Signal	Pins	I/O [1]	Description
Data			
D[0–31] Data Bus	32		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 MPC855T when it is external bus master and it initiated a write transaction to a slave device. For single-beat transactions, the byte lanes not selected for the transfer by the A[30–31] and TSIZ[0–1] will not supply valid data.
		I	Driven by the slave in a read transaction. For single-beat transactions, the byte lanes not selected for the transfer by the A[30–31] and TSIZ[0–1] will not be sampled by the MPC855T
DP[0–3] Parity Bus	4		Each parity line corresponds to each one of the data bus lanes: Data Bus ByteParity Line D[0–7]DP0 D[8–15]DP1 D[16–23]DP2 D[24–31]DP3
		O	Driven by the MPC855T when it is external bus master and it initiated a write transaction to a slave device. Each line has the parity value (even or odd) of its corresponding data bus byte. For single-beat transfers, byte lanes not selected by A[30–31] and TSIZ[0–1] will not have a valid parity line.
		I	Driven by the slave in a read transaction. Each parity line is sampled by the MPC855T and checked (if enabled) against the expected value parity value (even or odd) of its corresponding data bus byte. For single-beat transfers, byte lanes not selected by A[30–31] and TSIZ[0–1] are not sampled by the MPC855T and its parity lines will not be checked.
Transfer Cycle Termination			
$\overline{\text{TA}}$ Transfer Acknowledge	1	I	Driven by the slave device to which the current transaction is addressed. Indicates that the slave received the data on the write cycle or returned data on the read cycle. If the transaction is a burst, $\overline{\text{TA}}$ should be asserted for each beat.
		O	Driven by the MPC855T when the slave device is controlled by the on-chip memory controller or PCMCIA interface.
$\overline{\text{TEA}}$ Transfer Error Acknowledge	1	I	Driven by the slave device to which the current transaction is addressed. Indicates that an error condition occurred during the bus cycle.
		O	Driven by the MPC855T when the internal bus monitor detects a bus error.
$\overline{\text{BI}}$ Burst Inhibit	1	I	Driven by the slave device to which the current transaction was addressed. Indicates that the current slave does not support burst mode.
		O	Driven by the MPC855T when the on-chip memory controller controls the slave.

Table 13-1. MPC855T Signal Overview (continued)

Signal	Pins	I/O [1]	Description
Arbitration			
$\overline{BR}$ Bus Request	1	I	Asserting $\overline{BR}$ when the internal arbiter is enabled indicates an external master is requesting the bus.
		O	The MPC855T drives $\overline{BR}$ when the internal arbiter is disabled.
$\overline{BG}$ Bus Grant	1	O	When the internal arbiter is enabled, the MPC855T asserts $\overline{BG}$ to indicate that an external master may assume bus mastership and begin a bus transaction. The device requesting bus mastership should qualify $\overline{BG}$ to ensure it is the bus owner: Qualified $\overline{BG}$ = $\overline{BG}$ & ~ $\overline{BB}$
		I	When the internal arbiter is disabled, $\overline{BG}$ is sampled and properly qualified by the MPC855T when an external bus transaction is to be executed by the chip.
$\overline{BB}$ Bus Busy	1	O	When the internal arbiter is enabled, the MPC855T asserts $\overline{BB}$ to indicate it is bus master. When the internal arbiter is disabled, the MPC855T asserts $\overline{BB}$ after the external arbiter granted mastership to the chip and it is ready to start the transfer.
		I	When the internal arbiter is enabled, the MPC855T samples this signal to get 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.

[1] O= Output from the MPC855T; I= Input to the MPC855T

13.4 Bus Operations

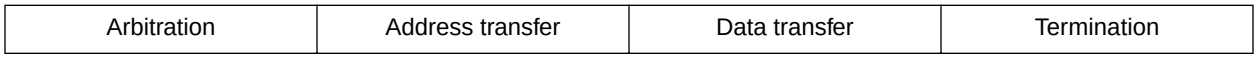
This section provides a functional description of the system bus, the signals that control it, and the bus cycles provided for data transfers. It also describes error conditions, bus arbitration, and the reset operation. The MPC855T generates a system clock output (CLKOUT), which directly sets the bus interface operation frequency. Internally, the MPC855T uses a phase-lock loop (PLL) circuit to generate a master clock for all core circuitry (including the bus interface), which is phase-locked to CLKOUT.

MPC855T bus interface signals 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. Because the same clock edge is used for driving or sampling bus signals, clock skew may occur between various modules in a system due to routing or the use of multiple clock lines. The system must handle any clock skew problems that could occur as a result of layout, lead length, and physical routing.

13.4.1 Basic Transfer Protocol

The basic transfer protocol defines the sequence of actions required for a complete MPC855T bus transaction. Figure 13-3 shows a simplification of the basic transfer protocol.

Figure 13-3. Basic Transfer Protocol



- Arbitration—A device requests bus access
- Address phase—The address and the transfer attributes are generated.
- Data phase—Any data to be transferred is transferred. The data phase may transfer a single beat of data (4 bytes or less) for nonburst operations, a 4-beat data burst (4 × 4 bytes), an 8-beat data burst (8 × 2 bytes), or a 16-beat data burst (16 × 1 bytes).
- Termination—The transfer completes successfully or it was aborted.

13.4.2 Single-Beat Transfer

During the data transfer, the master writes data to the slave or reads data from the slave. On a write cycle, the master drives the data as soon as it can, but not before the cycle after the address transfer phase. The master must consider the one dead clock cycle switching between drivers to avoid electrical contention. The master can stop driving the data bus as soon as it samples $\overline{TA}$ asserted on the rising edge of CLKOUT. On a read cycle the master accepts the data bus contents as valid at the rising edge of CLKOUT in which $\overline{TA}$ 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, then the data transfer. The following flow and timing diagrams show the handshakes applicable to the fixed transaction protocol.

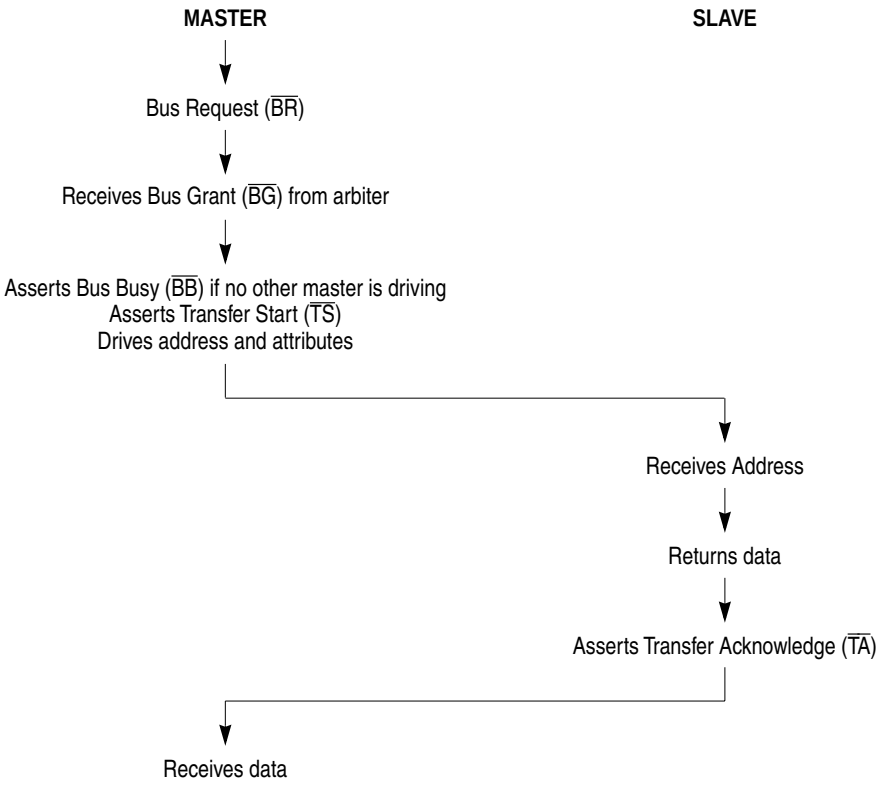


Figure 13-4. Basic Flow Diagram of a Single-Beat Read Cycle

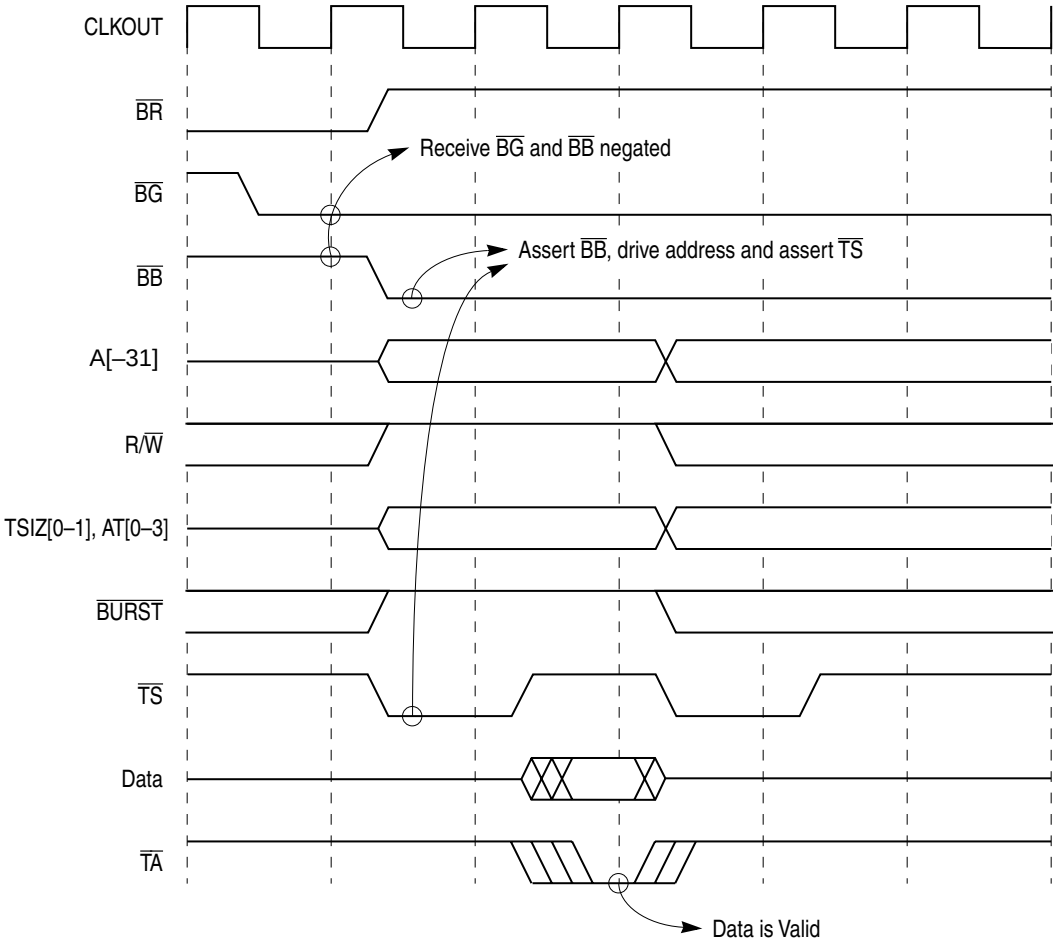


Figure 13-5. Basic Timing: Single-Beat Read Cycle, Zero Wait States

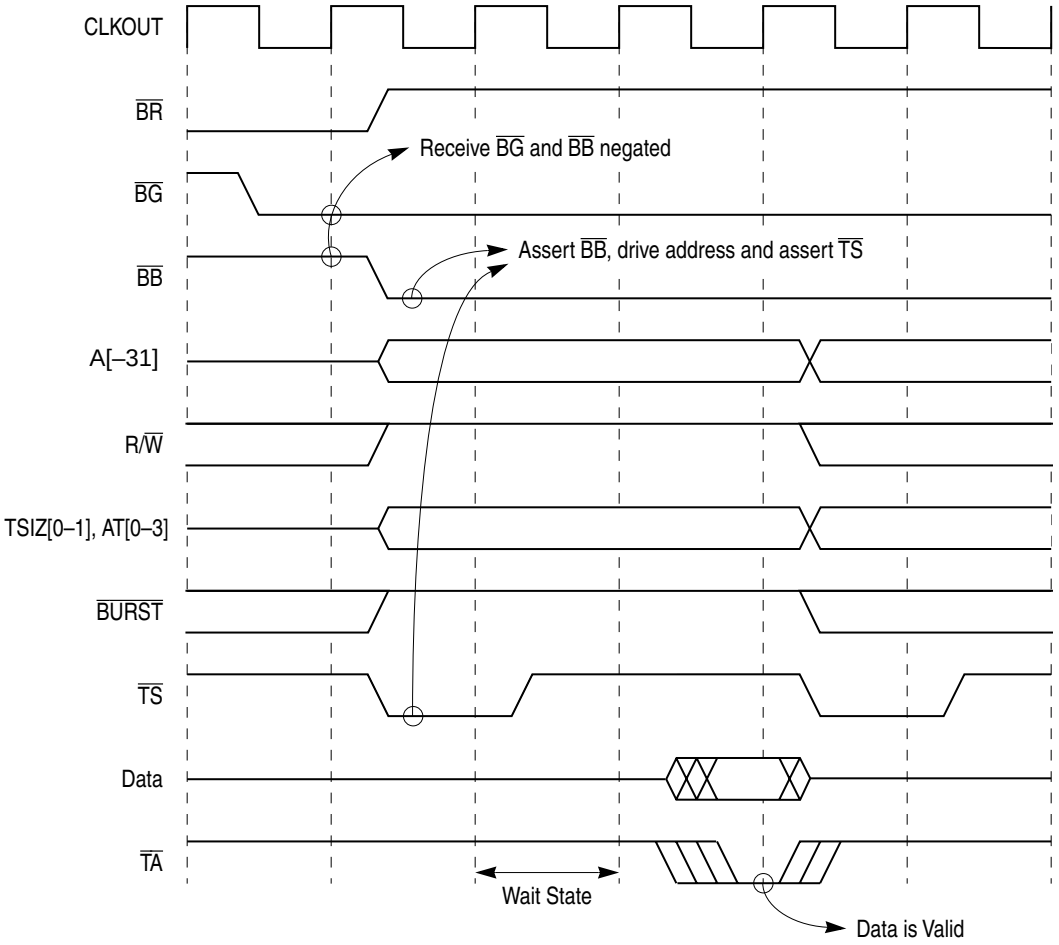


Figure 13-6. Basic Timing: Single-Beat Read Cycle, One Wait State

13.4.2.2 Single-Beat Write Flow

The basic write cycle begins with a bus arbitration, followed by the address transfer, then the data transfer. The following flow and timing diagrams show the handshakes as applicable to the fixed transaction protocol.

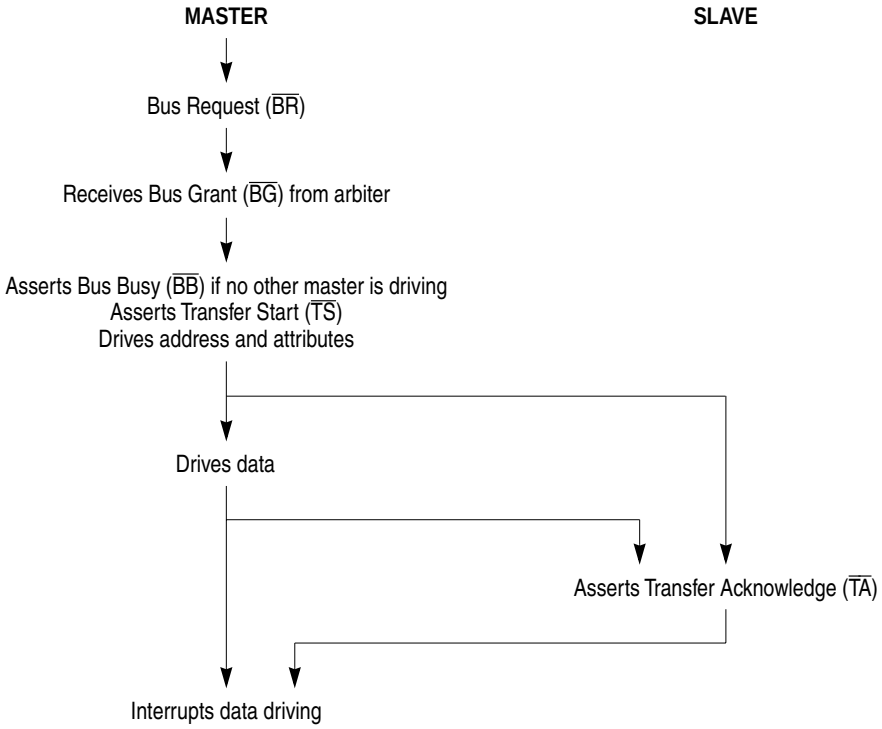


Figure 13-7. Basic Flow of a Single-Beat Write Cycle

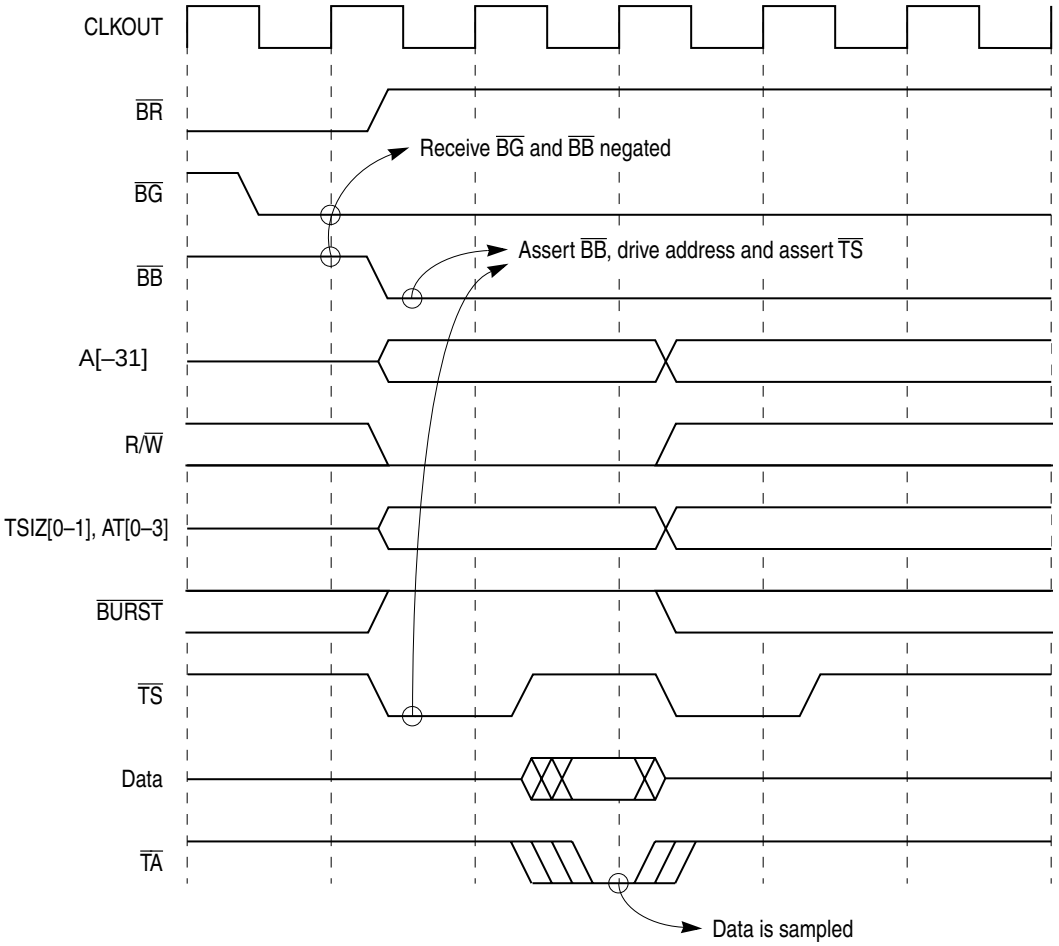


Figure 13-8. Basic Timing: Single-Beat Write Cycle, Zero Wait States

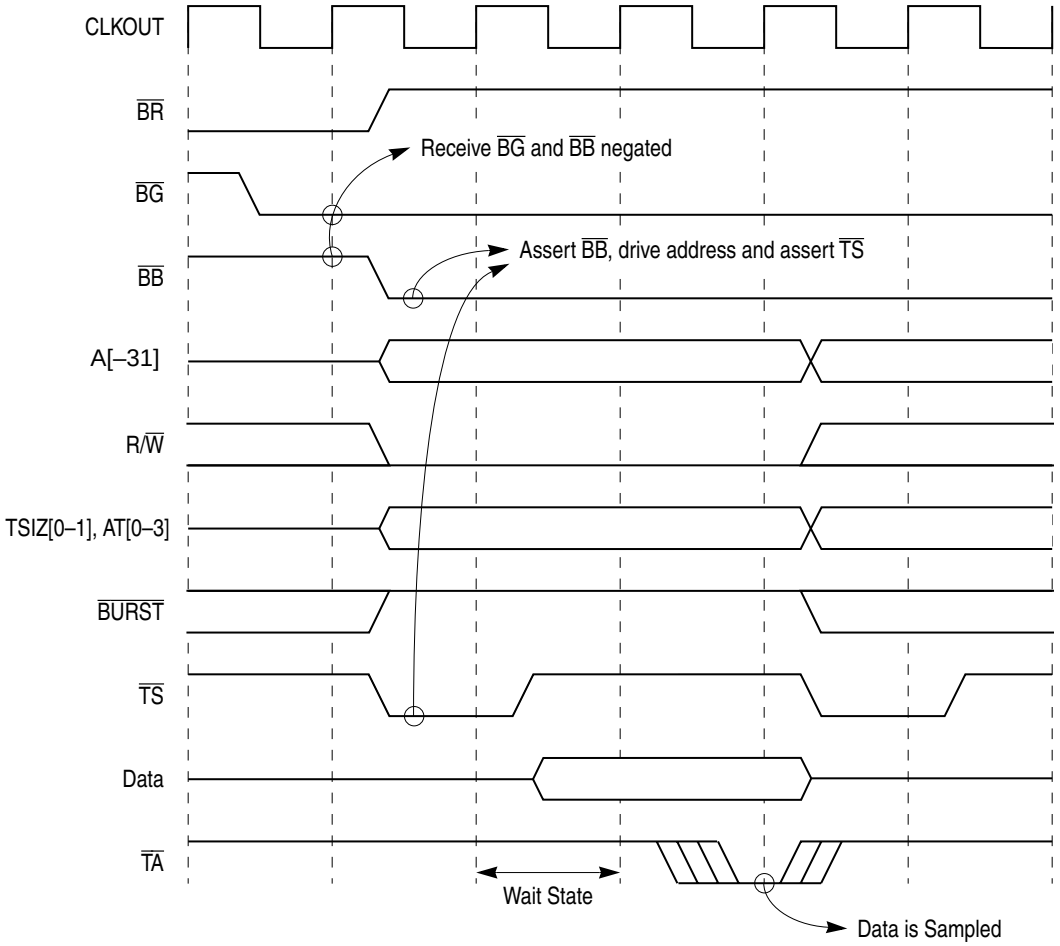


Figure 13-9. Basic Timing: Single-Beat Write Cycle, One Wait State

The general case of single-beat transfers assumes that external memory has a 32-bit port size. The MPC855T provides an effective mechanism for interfacing with 16- and 8-bit port size memories by allowing transfers to these devices when they are controlled by the internal memory controller.

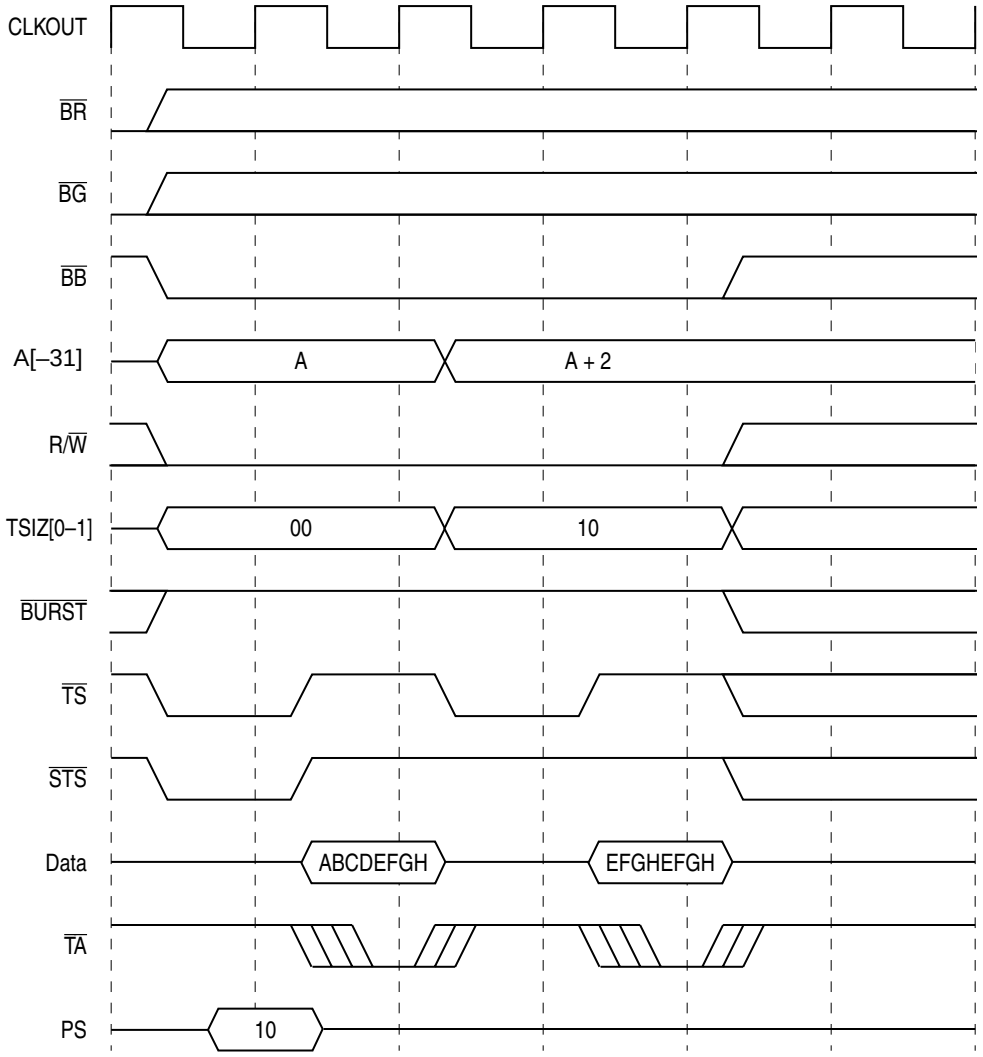


Figure 13-10. Basic Timing: Single-Beat, 32-Bit Data Write Cycle, 16-Bit Port Size

13.4.3 Burst Transfers

The MPC855T or other synchronous external bus devices use burst transfers to access 16-byte operands. A burst accesses a 16-byte block aligned to a 16-byte memory boundary by supplying a starting address that points to one of the words and requiring the memory device to sequentially drive/sample each word on the data bus. The selected slave device must internally increment A28 and A29 (and A30 in the case of a 16-bit port size slave device) of the supplied address for each transfer, causing the address to wrap around at the end of the four-word block. For slaves controlled by the memory controller, the MPC855T increments the address on A[28–31] and/or BADDR[28–30].

Address and transfer attributes supplied by the master bus remain stable during the transfers; the selected device terminates each transfer by asserting $\overline{TA}$ after each word transferred on the data bus. The MPC855T also supports burst-inhibited transfers for slave

devices that do not support bursting. For this type of cycle, the selected slave device supplies/samples the address of the first word of the burst and asserts the burst-inhibit signal ($\overline{BI}$) with $\overline{TA}$ for the first transfer of the burst access. The MPC855T responds by terminating the burst and accessing the rest of the 16-byte block, using three read/write cycles (each one for a word) for a 32-bit port-width slave, seven read/write cycles for a 16-bit port-width slave, or fifteen read/write cycles for a 8-bit port-width slave.

The general case of burst transfers assumes that external memory has a 32-bit port size. The MPC855T provides an effective mechanism for interfacing with 16-bit port size memories and 8-bit port size memories allowing burst transfers to these devices when they are controlled by the internal memory controller. In this case, the MPC855T attempts to initiate a burst transfer as in the normal case. If, in a cycle before the $\overline{TA}$ is asserted for the first beat, the memory controller responds that the port size is 16-/8-bits and that the burst is accepted, the MPC855T completes a 8-/16-beat burst. Each data beat effectively transfers only 2/1 bytes. Note that this 8-/16-beat burst is considered an atomic transaction, so the MPC855T will not allow other unrelated master accesses or bus arbitration between transfers.

13.4.4 Burst Operations

The MPC855T burst mechanism uses additional signals to the basic protocol: $\overline{BURST}$ indicates that the cycle is a burst cycle, burst data in progress ($\overline{BDIP}$) indicates the duration of the burst data, and burst inhibit ($\overline{BI}$) indicates whether the slave supports bursts. Along with asserting $\overline{TS}$, the master drives the address, address attributes, and $\overline{BURST}$ signals to indicate that a burst transfer is being initiated. Slaves that support bursting negate $\overline{BI}$. If the slave cannot burst, it asserts $\overline{BI}$. During the data phase of a burst write cycle the master drives the data. The master also asserts $\overline{BDIP}$ if it intends to drive the data beat after the current one.

When the slave has received the data, it asserts $\overline{TA}$ to indicate to the master that it is ready for the next transfer. The master again drives the next data and asserts or negates $\overline{BDIP}$. If the master does not intend to drive another data beat, it negates $\overline{BDIP}$ to indicate to the slave that the next data beat is the last one in the burst write.

Bursts performed by MPC855T internal masters are always 16 bytes. The MPC855T memory controller responds only to fixed-length bursts (also typically programmed to be 16 bytes). Therefore, devices in an MPC855T system should attempt only 16-byte burst transfers except for external masters with a dedicated chip select, such as an external MPC603 that bursts to a chip select programmed for a 32-byte burst.

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 $\overline{BDIP}$. When the master receives the next-to-last data, it negates $\overline{BDIP}$. Thus, the slave stops driving new data after receiving the negation of $\overline{BDIP}$ at the rising clock edge.

In the case of 32-bit port size, the burst includes 4 beats. When the port size is 16 bits and controlled by the internal memory controller, the burst includes 8 beats. When the port size is 8 bits and controlled by the internal memory controller, the burst includes 16 beats. The MPC855T bus supports critical data first access for fixed-size burst. The order of wraparound wraps back to the critical data. For example, assuming data 2 is critical:

- Case burst of four:

  data 2 → data 3 → data 0 → data 1
- Case burst of eight:

data 2 → data 3 → data 4 →..... → data 7 → data 0 → data 1

The following flow and timing diagrams show the handshakes for burst transactions.

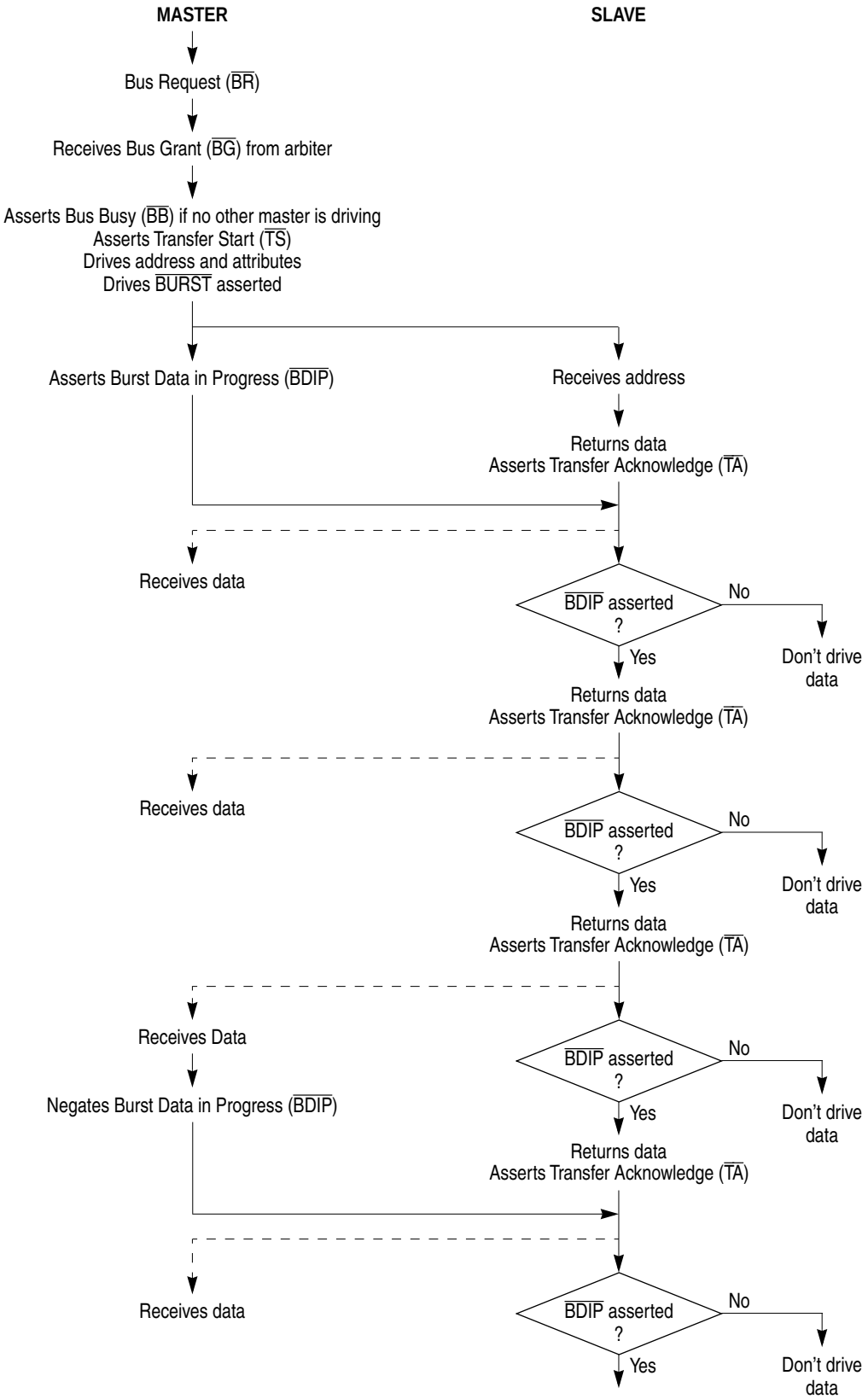


Figure 13-11. Basic Flow of a Burst-Read Cycle

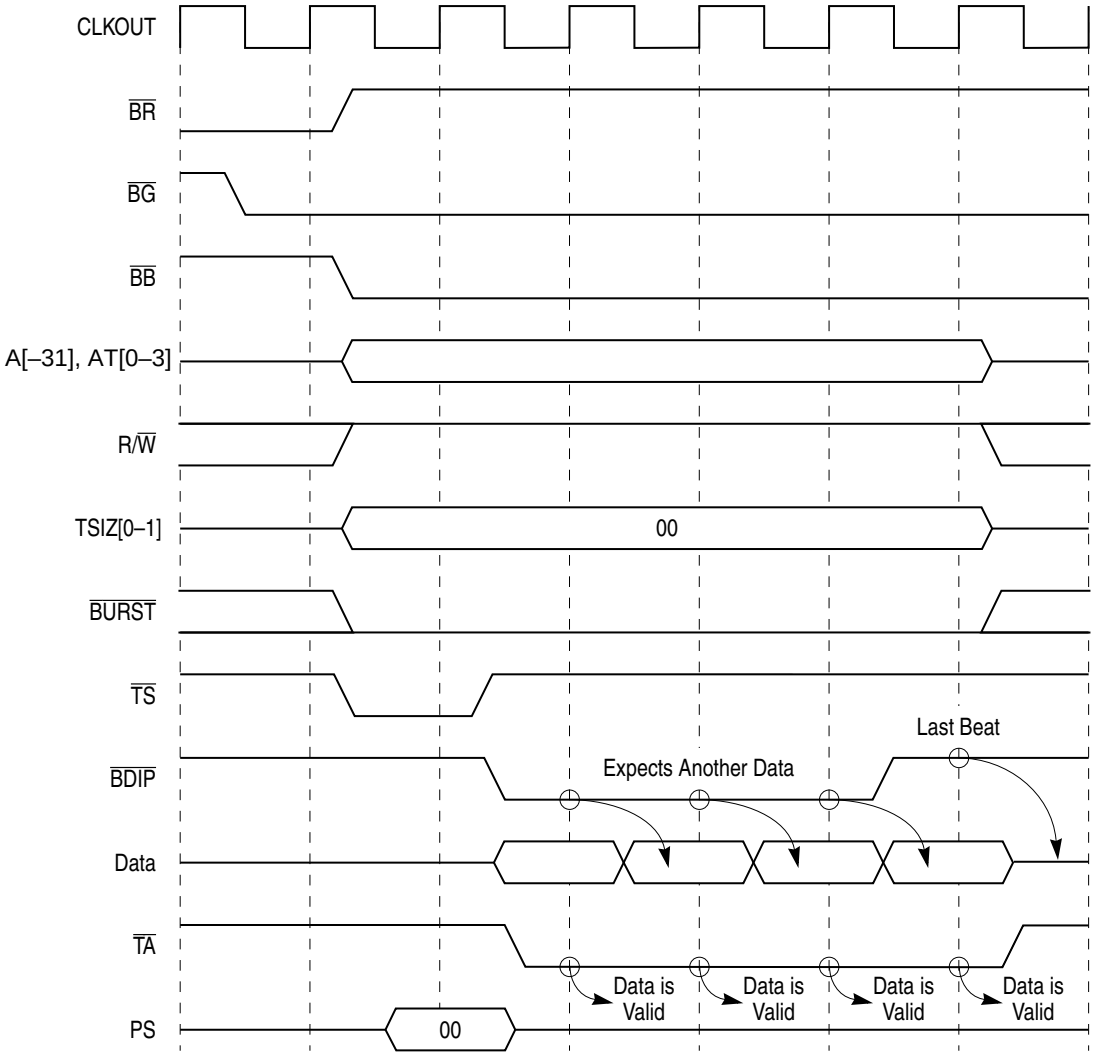


Figure 13-12. Burst-Read Cycle: 32-Bit Port Size, Zero Wait State

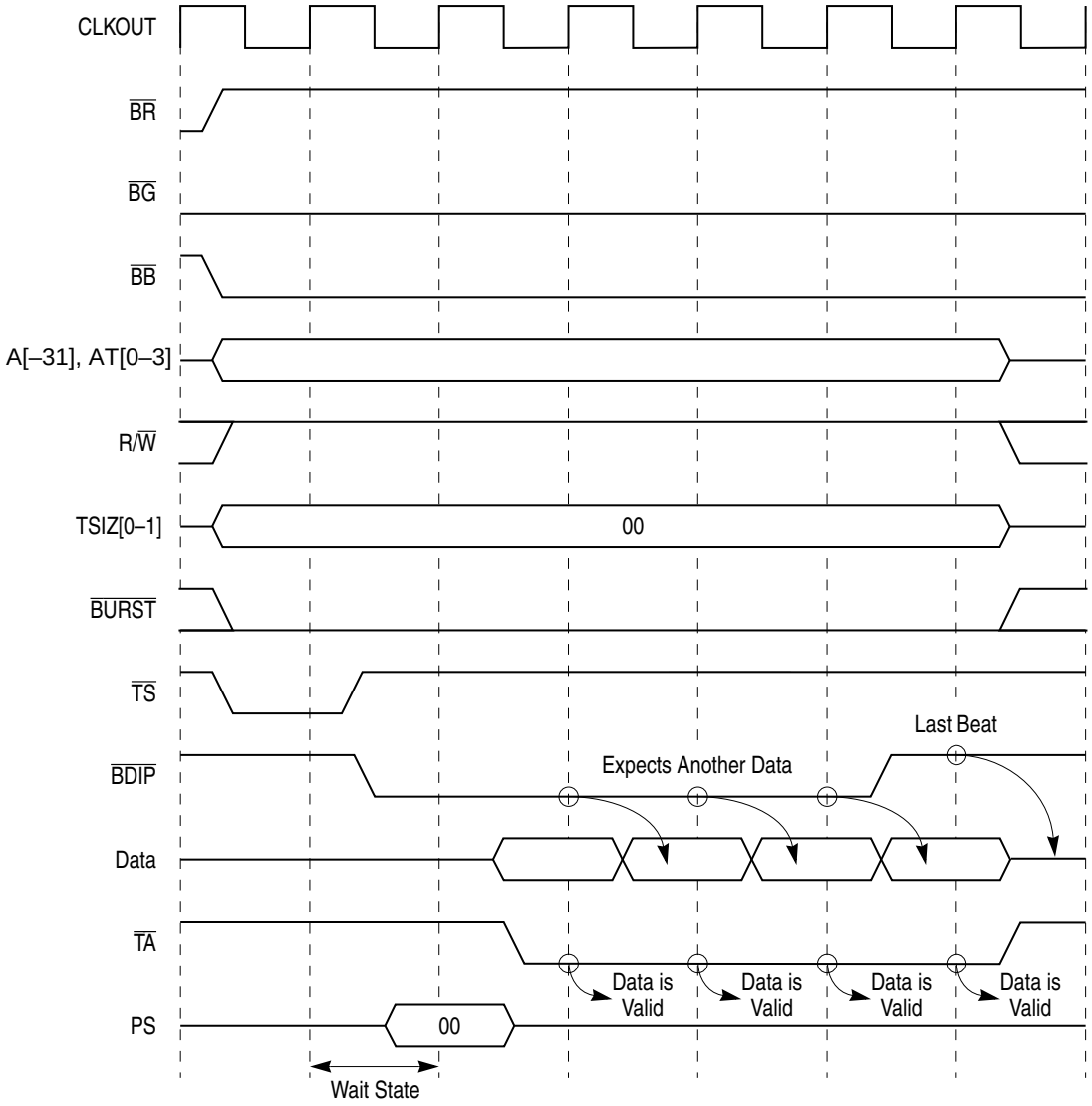


Figure 13-13. Burst-Read Cycle: 32-Bit Port Size, One Wait State

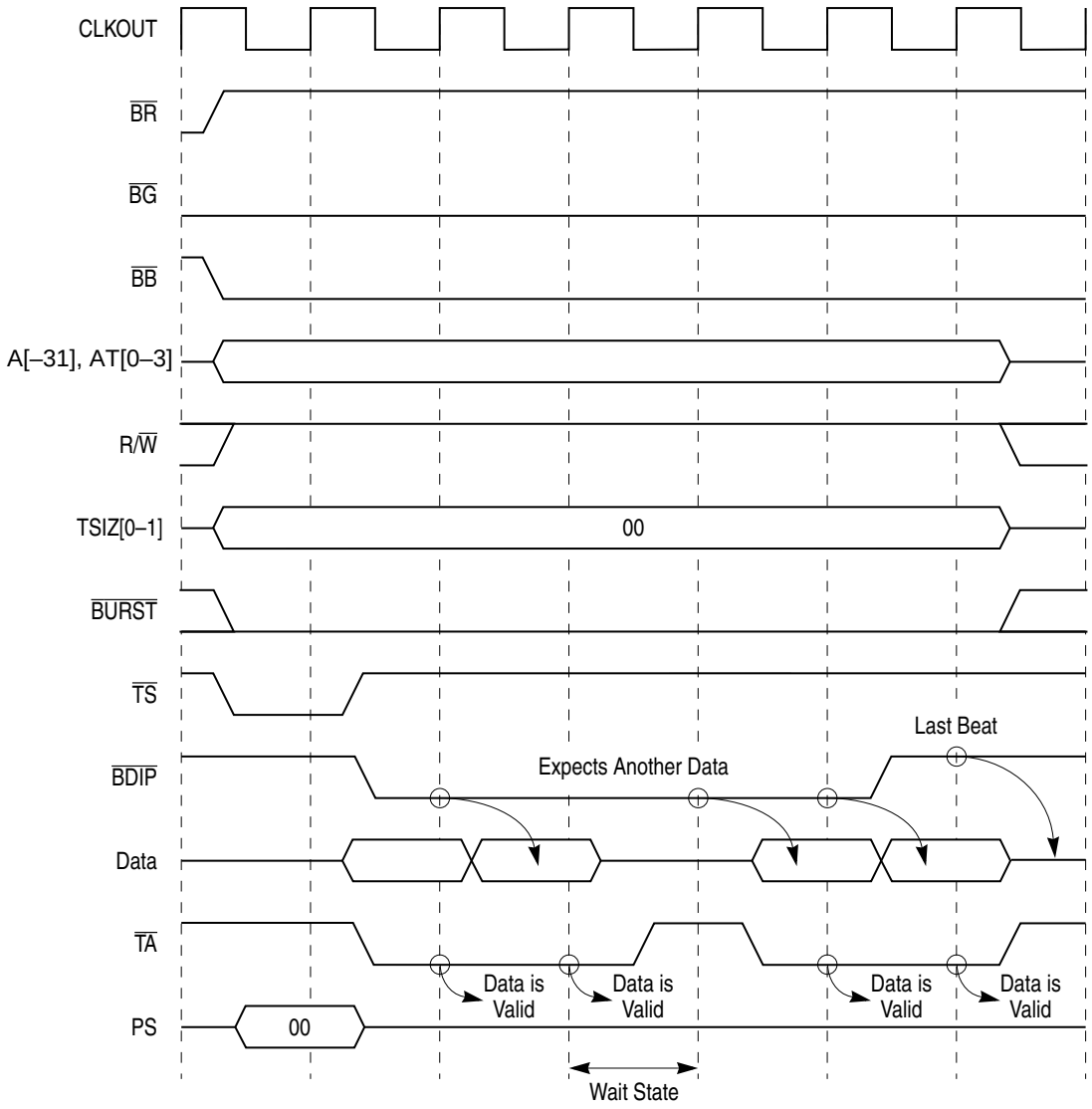


Figure 13-14. Burst-Read Cycle: 32-Bit Port Size, Wait States between Beats

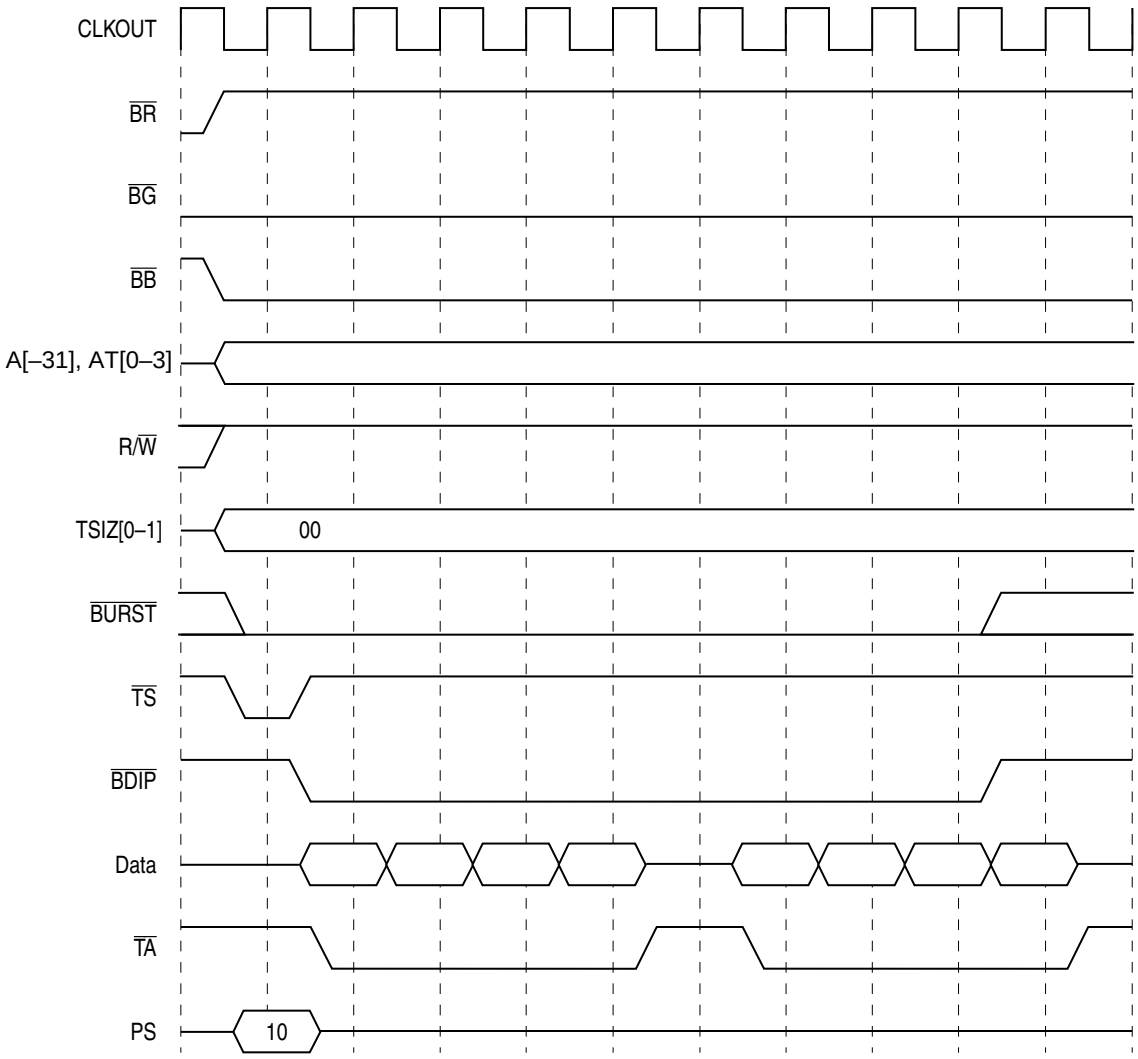


Figure 13-15. Burst-Read Cycle: 16-Bit Port Size, One Wait State between Beats

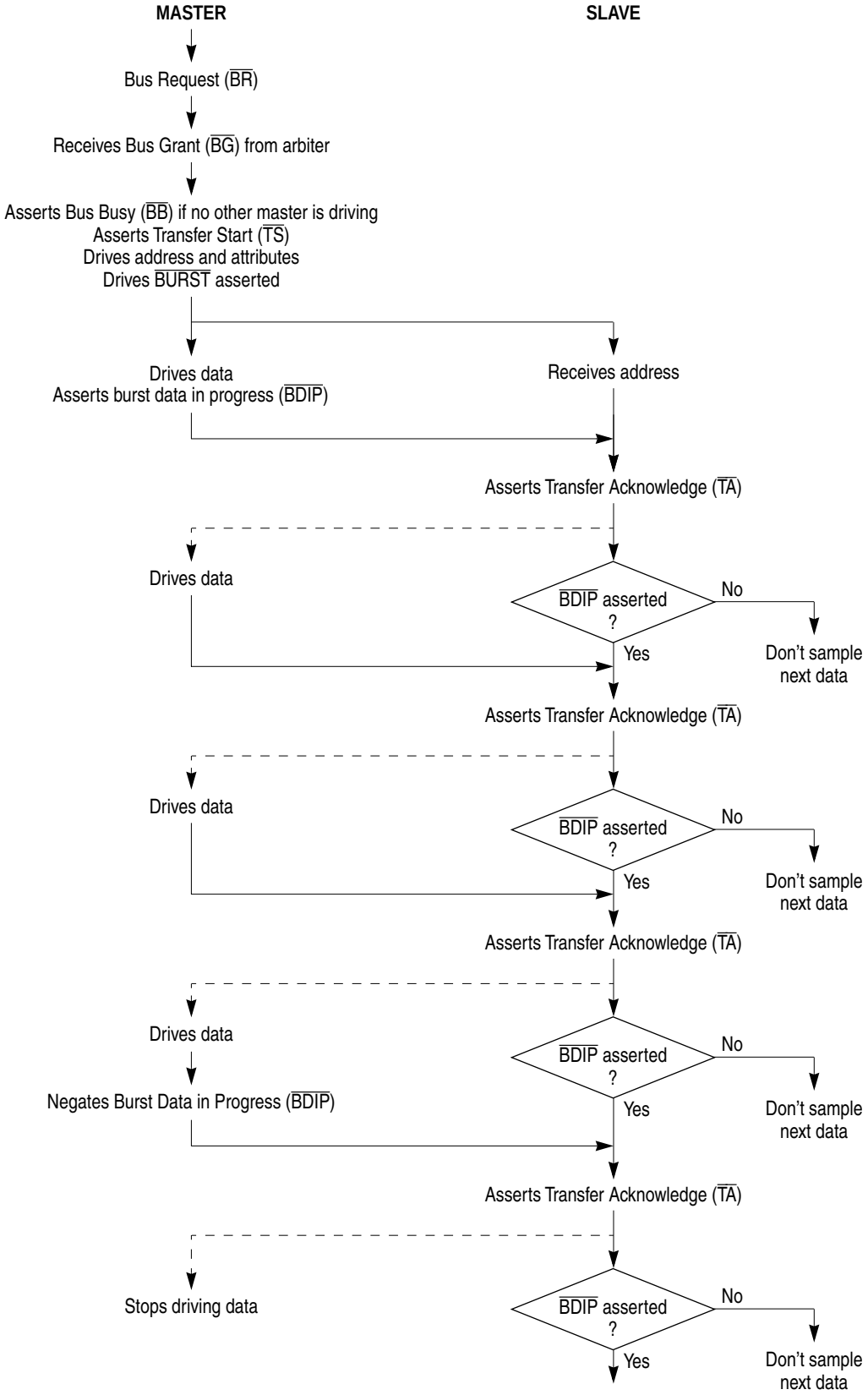


Figure 13-16. Basic Flow of a Burst Write Cycle

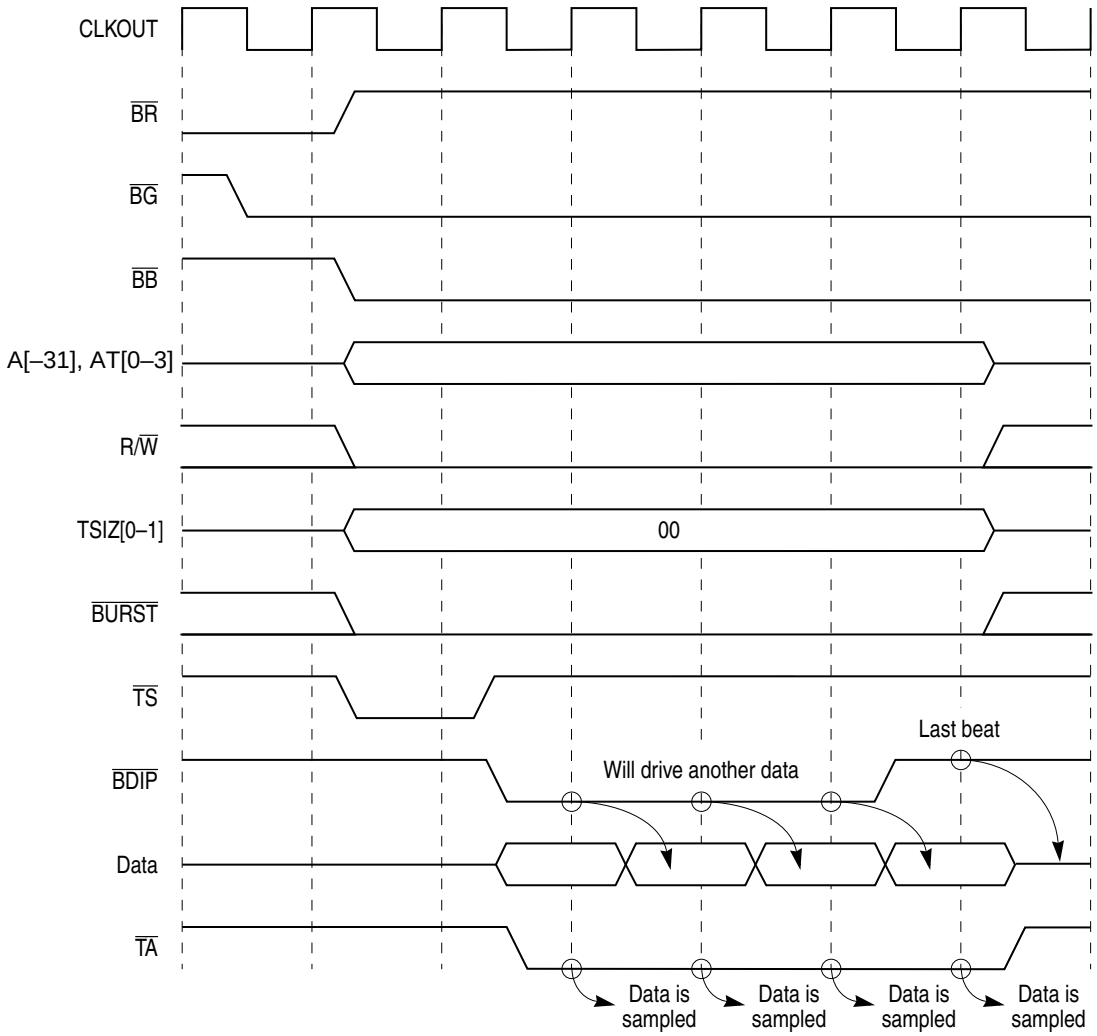


Figure 13-17. Burst-Write Cycle: 32-Bit Port Size, Zero Wait States

Figure 13-18 shows an attempted burst read to a slave device that does not support bursting. The slave acknowledges the first transfer and also asserts the burst-inhibit signal ($\overline{BI}$). The MPC855T responds by terminating the burst and accessing the rest of the 16-byte block, using three single-beat read cycles.

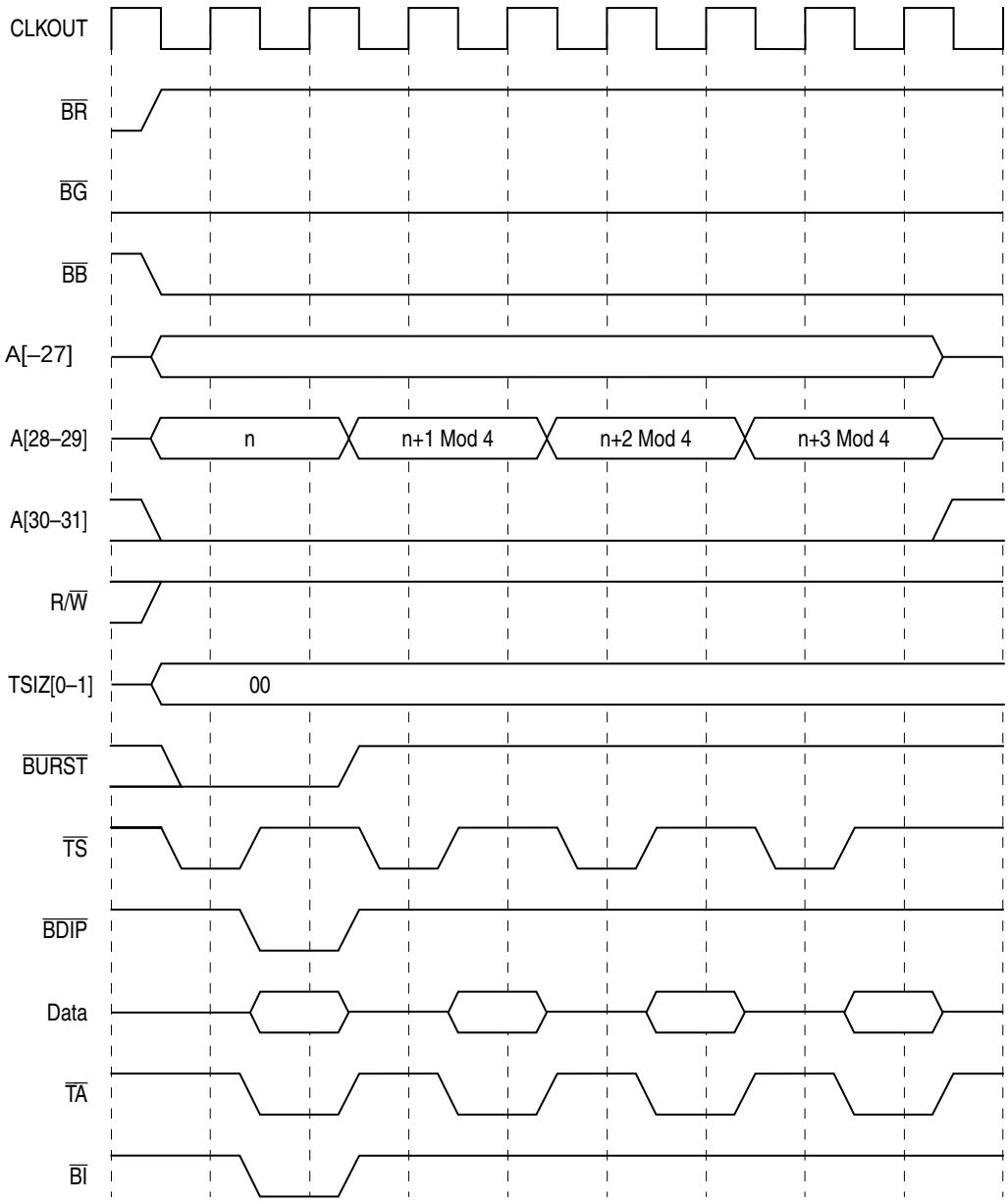


Figure 13-18. Burst-Inhibit Cycle: 32-Bit Port Size

13.4.5 Alignment and Data Packing on Transfers

The MPC855T external bus supports only natural address alignment:

- Byte access can have any address alignment
- Half-word access must have A[31] = 0b0
- Word access must have A[30–31] = 0b00
- For burst accesses A[30–31] = 0b00

Misaligned accesses performed by the core are broken into multiple bus accesses with natural alignment. Misaligned accesses performed by external masters are not supported.

The MPC855T transfers operands through its 32-bit data port. If the transfer is controlled by the internal memory controller, the MPC855T 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 be fixed. A 32-bit port must reside on D[0–31], a 16-bit port must reside on D[0–15], and an 8-bit port must reside on D[0–7]. The MPC855T always tries to transfer the maximum amount of data on all bus cycles; for a word operation, it always assumes that the port is 32 bits wide when beginning the cycle. Figure 13-19, Figure 13-20, Table 13-2, and Table 13-3 use the following conventions:

- OP0 is the MSB of a word operand; OP3 is the LSB.
- The two bytes of a half-word operand are OP0 (most-significant) and OP1 or OP2 (most-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.

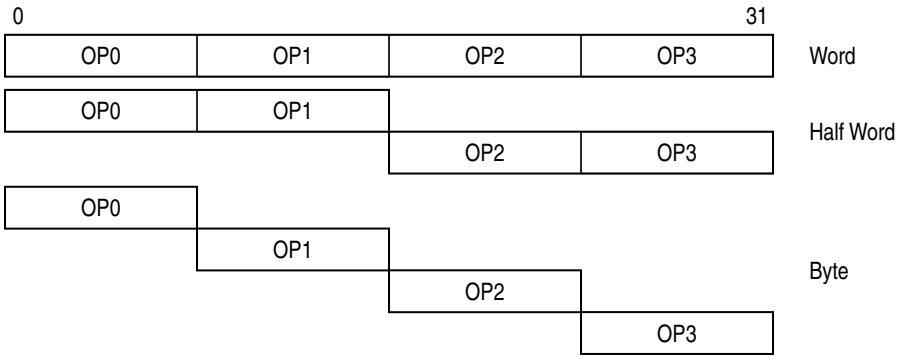


Figure 13-19. Internal Operand Representation

Figure 13-20 shows the device connections on the data bus.

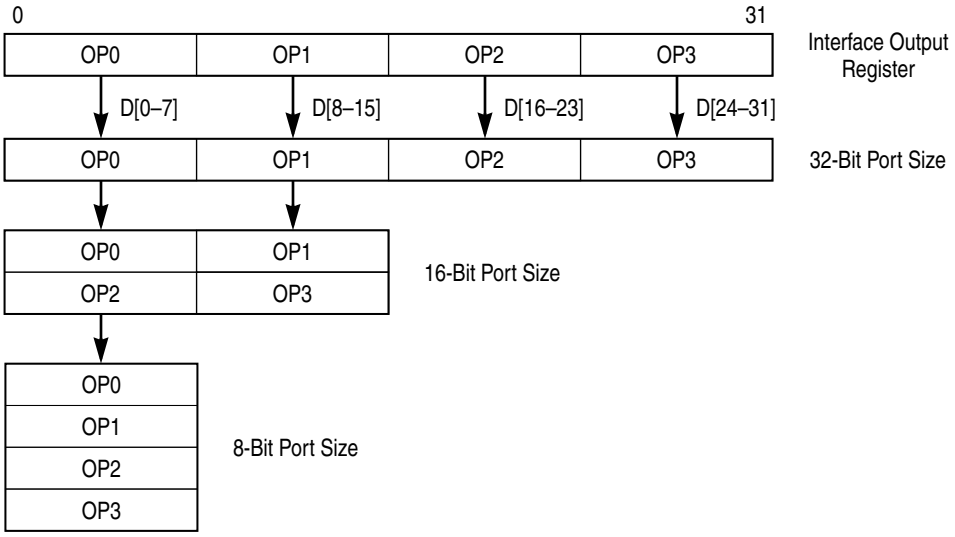


Figure 13-20. Interface to Different Port Size Devices

Table 13-2 lists the bytes required on the data bus for read cycles.

Table 13-2. Data Bus Requirements for Read Cycles

Transfer Size	TSIZ[0–1]		Address		32-Bit Port [1]				16-Bit Port[1]		8-Bit Port
			A30	A31	D[0–7]	D[8–15]	D[16–D3]	D[24–31]	D[0–7]	D[8–15]	D[0–7]
Byte	0	1	0	0	OP0	—	—	—	OP0	—	OP0
			0	1	—	OP1	—	—	—	OP1	OP1
			1	0	—	—	OP2	—	OP2	—	OP2
			1	1	—	—	—	OP3	—	OP3	OP3
Half-Word	1	0	0	0	OP0	OP1	—	—	OP0	OP1	OP0
			1	0	—	—	OP2	OP3	OP2	OP3	OP2
Word	0	0	0	0	OP0	OP1	OP2	OP3	OP0	OP1	OP0

1 — denotes a byte not required during that read cycle.

Table 13-3 lists data transfer patterns for write cycles when the MPC855T initiates accesses.

Table 13-3. Data Bus Contents for Write Cycles

Transfer Size	TSIZ[0–1]		Address		External Data Bus Pattern [1]			
			A30	A31	D[0–7]	D[8–15]	D[16–D3]	D[24–31]
Byte	0	1	0	0	OP0	—	—	—
			0	1	OP1	OP1	—	—
			1	0	OP2	—	OP2	—
			1	1	OP3	OP3	—	OP3
Half-Word	1	0	0	0	OP0	OP1	—	—
			1	0	OP2	OP3	OP2	OP3
Word	0	0	0	0	OP0	OP1	OP2	OP3

[1] — denotes a byte not required during that read cycle.

13.4.6 Arbitration Phase

The external bus design provides for a single bus master at any one time, either the MPC855T or an external device. The arbitration of external bus devices contending for bus mastership 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 MPC855T. The arbitration configuration (external or internal) is set at system reset. See Section 15.8, “External Master Support.”

Each bus master must have bus request ($\overline{BR}$), bus grant ($\overline{BG}$), and bus busy ($\overline{BB}$) signals. A device needing the bus asserts $\overline{BR}$, and then waits for the arbiter to assert $\overline{BG}$. The new master must look at $\overline{BB}$ to ensure that no other master is driving the bus before it can assert $\overline{BB}$ to assume bus mastership. (Note that the internal arbiter may take away the $\overline{BG}$ if an internal master of higher priority requests the bus and the new master does not assert $\overline{BB}$ within one clock after $\overline{BG}$.)

If the arbiter removes the bus grant from a device that wants another transfer, the device must rearbitrate for bus mastership. The MPC855T, however, guarantees data coherency for accesses to small ports and for decomposed bursts. This means that the MPC855T does not release the bus before atomic transactions complete. For example, a halfword transfer to a byte port is broken into two byte transfers; the MPC855T does not deassert $\overline{BB}$ until the second transfer completes, unless an error occurs. Figure 13-21 shows basic bus arbitration protocol. Section 10.4.2, “SIU Module Configuration Register (SIUMCR),” describes how prioritization can be configured.

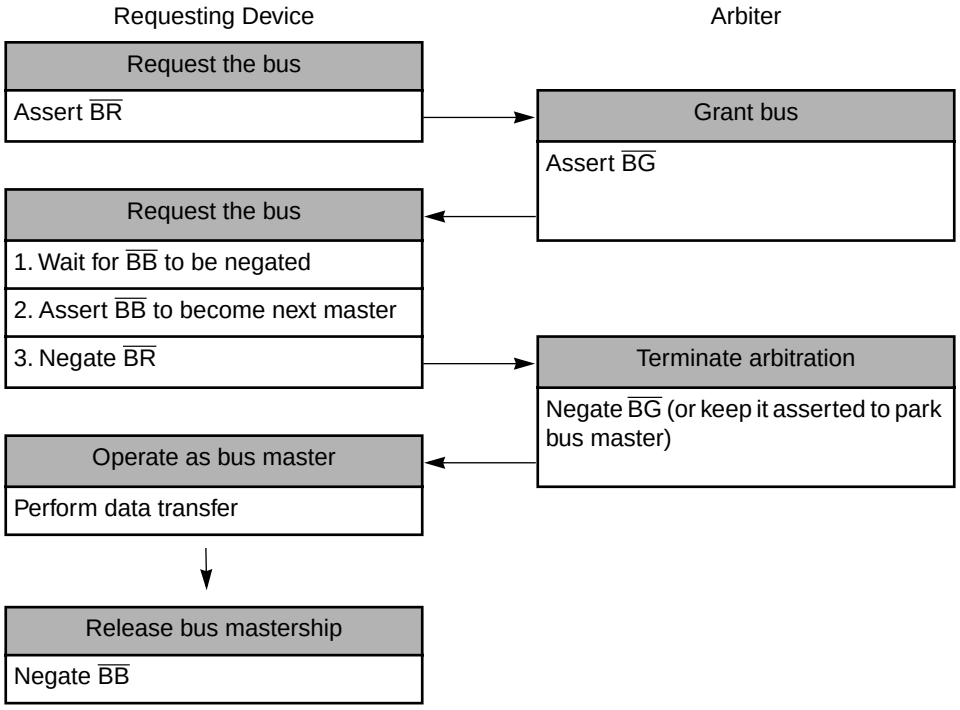


Figure 13-21. Basic Bus Arbitration Protocol

13.4.6.1 Bus Request (BR)

The potential bus master asserts BR to request bus mastership. BR should be negated as soon as the bus is granted, the bus is not busy, and the new master can drive the bus. If requests are pending, the master can assert BR as long as needed. When configured for external arbitration, the MPC855T drives BR when it requires bus mastership. When the internal on-chip arbiter is used, BR is an input to the internal arbiter and should be driven by the external bus master.

13.4.6.2 Bus Grant (BG)

The arbiter asserts BG to indicate that the bus is granted to the requesting device. BG can be negated after BR is negated or it can remain asserted to park the current master on the bus. The internal arbiter may take away the BG if the new master does not assert BB within one clock.

When the internal on-chip arbiter is used, BG is an output from the internal arbiter to the external bus master. When configured for external central arbitration, BG is an input to the MPC855T from the external arbiter.

13.4.6.3 Bus Busy (BB)

BB indicates that the current master is using the bus. New masters should not begin a transfer until BB is deasserted. The bus master should not relinquish or negate BB until it

completes its transfer. To avoid contention on $\overline{BB}$, masters should three-state $\overline{BB}$ when it gets a logical 1 value. This situation implies an external pull-up resistor is needed to ensure that a master that acquires the bus can recognize the negation of $\overline{BB}$, regardless of how many cycles have passed since the previous master relinquished the bus. See Figure 13-22.

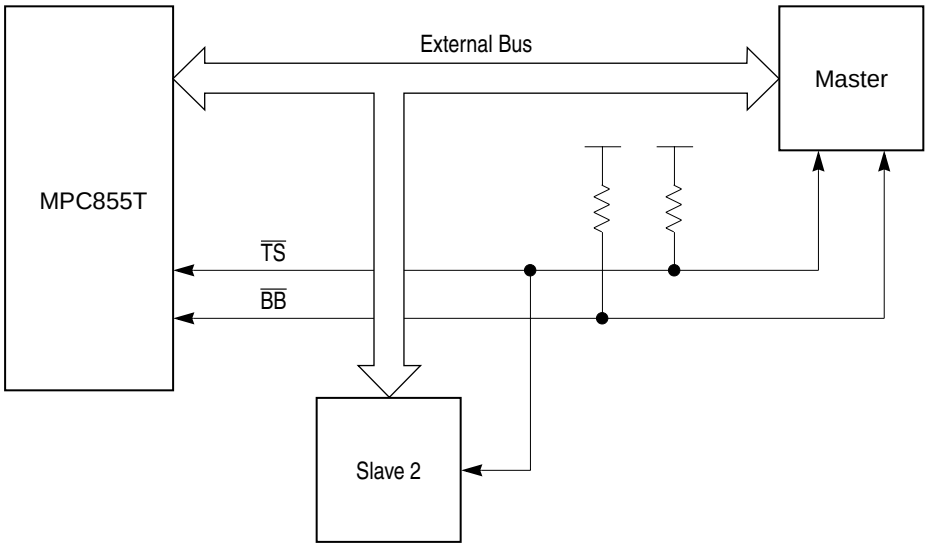


Figure 13-22. Bus Busy ($\overline{BB}$) and Transfer Start ($\overline{TS}$) Connection Example

Figure 13-23 shows an example bus arbitration between two contending masters.

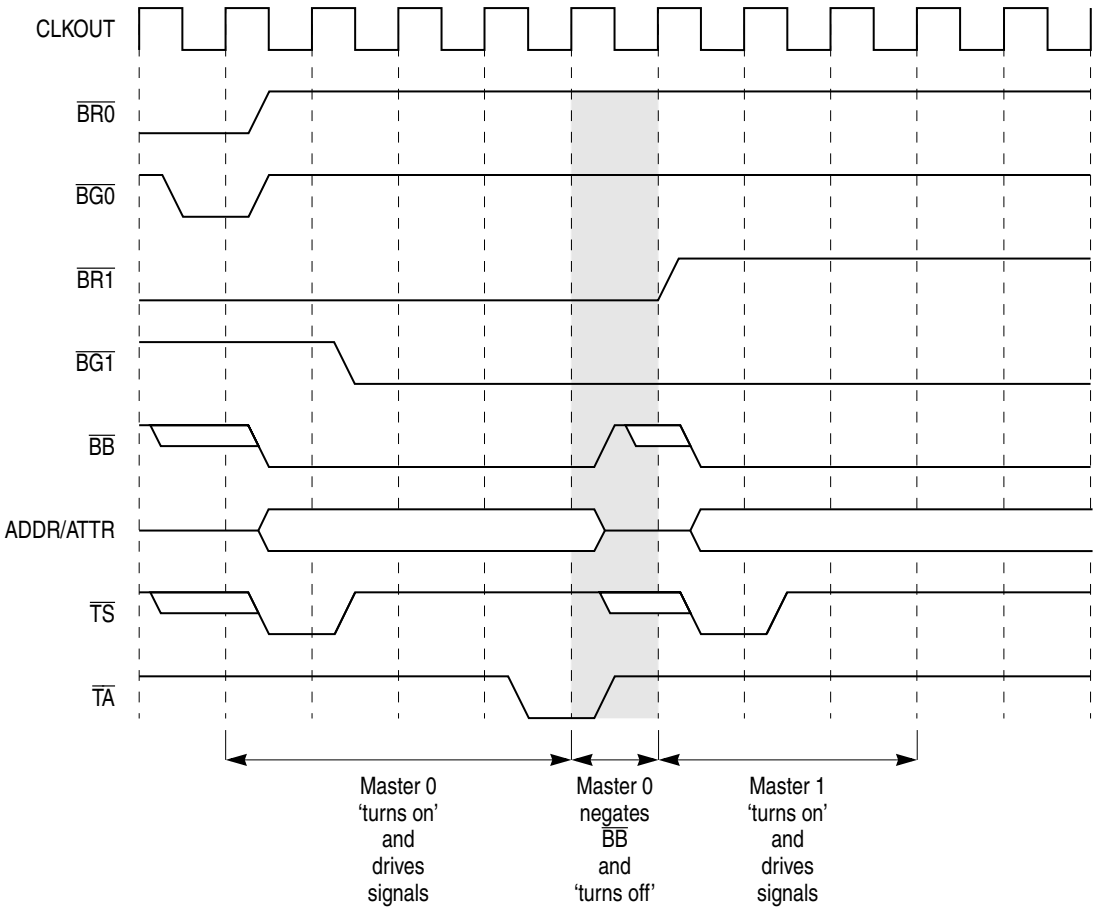


Figure 13-23. Bus Arbitration Timing Diagram

The MPC855T can be configured at system reset to use the internal bus arbiter. In this case, the MPC855T is parked on the bus. Section 10.4.2, "SIU Module Configuration Register (SIUMCR)," describes prioritization of external devices relative to the internal MPC855T bus masters. If the external device requests the bus and the MPC855T does not require it, or the external device has higher priority than the current internal bus master, the MPC855T grants the bus to the external device. Figure 13-24 shows the internal finite state machine that implements the arbiter protocol.

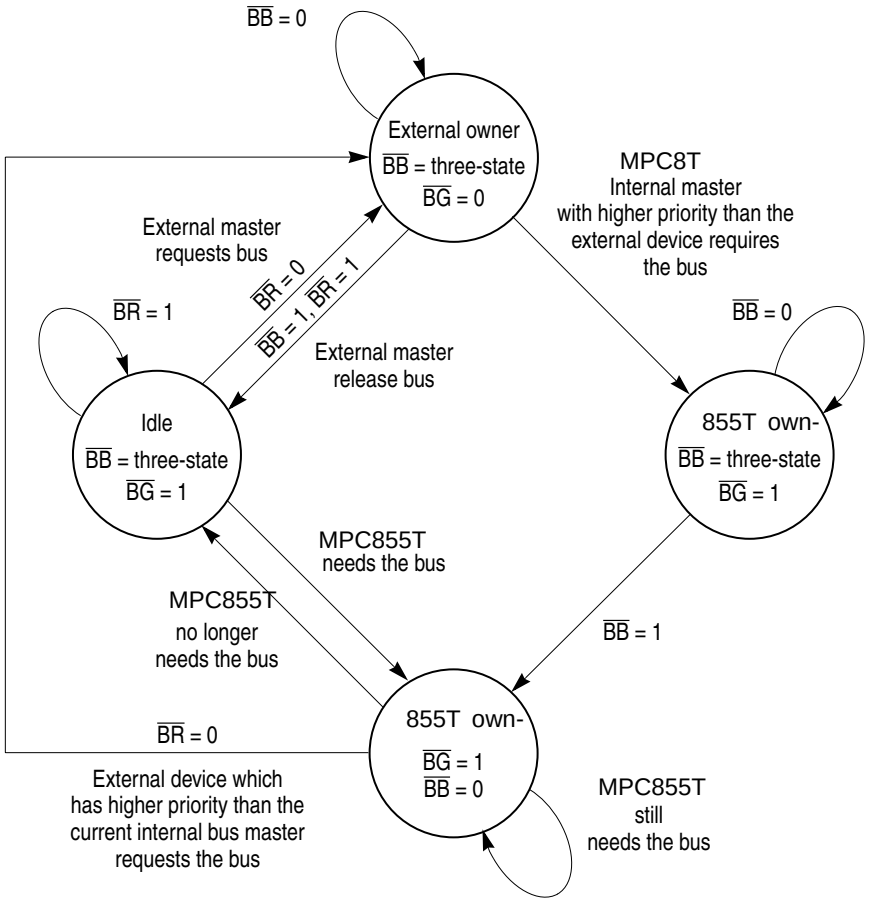


Figure 13-24. Internal Bus Arbitration State Machine

13.4.6.4 External Bus Parking

During external arbitration, the MPC855T supports bus parking. If the MPC855T detects that an external arbiter has asserted $\overline{BG}$ to it and $\overline{BB}$ is negated, the MPC855T starts a transfer without asserting $\overline{BR}$.

13.4.7 Address Transfer Phase-Related Signals

The following sections describe the address transfer phase-related signals.

13.4.7.1 Transfer Start ($\overline{TS}$)

The transfer start signal ($\overline{TS}$) indicates the beginning of a transaction on the bus addressing a slave device. A device should assert $\overline{TS}$ only after the arbitration protocol has granted mastership. $\overline{TS}$ is asserted only for the first cycle of the transaction and is negated in the successive clock cycles until the end of the transaction. To avoid contention, the master should three-state this signal when it relinquishes the bus. This situation indicates that an

external pull-up resistor should be connected to $\overline{TS}$ to avoid having a slave recognize this signal as asserted when no master drives it; see Figure 13-22.

13.4.7.2 Address Bus

The 32-bit address bus, A[0–31], is byte addressable, so each address can address one or more bytes. A[0] is the msb. The address and its attributes are driven on the bus with $\overline{TS}$; they remain valid until the bus master receives a transfer acknowledge from the slave. To distinguish an individual byte, the slave device must observe the TSIZ signals.

13.4.7.3 Transfer Attributes

The transfer attributes signal group consists of RD/$\overline{WR}$, $\overline{BURST}$, TSIZ[0–1], AT[0–3], $\overline{STS}$, and $\overline{BDIP}$. These signals (with the exception of the $\overline{BDIP}$) are available at the same time as the address bus.

13.4.7.3.1 Read/Write (RD/$\overline{WR}$)

RD/$\overline{WR}$ high indicates a read access and low indicates a write access. Driven at the beginning of a bus cycle, RD/$\overline{WR}$ is valid at the rising edge of the clock in which $\overline{TS}$ is asserted. RD/$\overline{WR}$ changes levels only when a write cycle is preceded by a read cycle or vice versa. It may remain low for consecutive write cycles.

13.4.7.3.2 Burst Indicator ($\overline{BURST}$)

$\overline{BURST}$ is driven by the bus master at the beginning of the bus cycle (along with the address) to indicate that the transfer is a burst transfer.

13.4.7.3.3 Transfer Size (TSIZ)

TSIZ[0–1] indicates the size of the requested data transfer. The TSIZ signals may be used with $\overline{BURST}$ and A[30–31] to determine which data byte lanes are used in the transfer. For nonburst transfers, TSIZ[0–1] specifies the number of bytes starting from the byte location addressed by A[30–31]. In burst transfers, the value of TSIZ[0–1] is always 00.

Table 13-4. $\overline{BURST}$/TSIZ Encoding

$\overline{BURST}$	TSIZ[0–1]	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 Types (AT)

The address type signals (AT[0–3]), $\overline{\text{PTR}}$ and $\overline{\text{RSV}}$, are outputs that indicate one of 16 address types to which the address applies. These types are designated as either a normal/alternate master cycle, user/supervisor (problem/privilege), and instruction/data types. The address type signals are valid at the rising edge of the clock in which the special transfer start ($\overline{\text{STS}}$) signal is asserted.

Address type 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 modified data cache block is copied back after the privilege level of the processor has been changed since the last access to the same cache block. A functional usage of the address type signal $\overline{\text{RSV}}$ is for the reservation protocol described in Section 13.4.9, “Memory Reservation.” Table 13-5 provides the space definition encoded by the $\overline{\text{STS}}$, $\overline{\text{TS}}$, AT[0–3], $\overline{\text{PTR}}$, and $\overline{\text{RSV}}$.

Table 13-5. Address Types Definition

$\overline{\text{STS}}$	$\overline{\text{TS}}$	Core/ CPM (AT0)	User/ Supervisor (AT1)	Instruction/ Data (AT2)	Reservation/ Program Trace (AT3)	Program Trace ($\overline{\text{PTR}}$)	Reservation ($\overline{\text{RSV}}$)	Address Space Definitions
1	x	x	x	x	x	1	1	No transfer or not the 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-initiated, normal instruction, program trace, supervisor mode
					1	1	1	Core-initiated, normal instruction, supervisor mode
				1	0	1	0	Core-initiated, reservation data, supervisor mode
					1	1	1	Core-initiated, normal data, supervisor mode
			1	0	0	0	1	Core-initiated, normal instruction, program trace, user mode
					1	1	1	Core-initiated, normal instruction, user mode
				1	0	1	0	Core-initiated, reservation data, user mode
					1	1	1	Core-initiated, normal data, user mode
		1	AT1	AT2	AT3	1	1	DMA-initiated, normal, AT[1–3] user-programmable (see IDMA and DMA function code registers)

Table 13-5. Address Types Definition (continued)

STS	TS	Core/ CPM (AT0)	User/ Supervisor (AT1)	Instruction/ Data (AT2)	Reservation/ Program Trace (AT3)	Program Trace (PTR)	Reservation (RSV)	Address Space Definitions
x	1	0	0	0	0	0	1	Core-initiated, show cycle address instruction, program trace, supervisor mode
					1	1	1	Core-initiated, show cycle address instruction, supervisor mode
				1	0	1	0	Core-initiated, reservation show cycle data, supervisor mode
					1	1	1	Core-initiated, show cycle data, supervisor mode
			1	0	0	0	1	Core-initiated, show cycle address instruction, program trace, user mode
					1	1	1	Core-initiated, show cycle address instruction, user mode
				1	0	1	0	Core-initiated, reservation show cycle data, user mode
					1	1	1	Core-initiated, show cycle data, user mode
		1	AT1	AT2	AT3	1	1	DMA-initiated, normal, AT[1–3] user-programmable (see IDMA and DMA function code registers)

Show cycles are accesses to the core's internal bus devices. These accesses are made visible externally for emulation and debugging. 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 a read access. The address of the show cycle is valid on the bus for one clock and the data of the show cycle is valid on the bus for one clock. The data phase does not require a transfer acknowledge to terminate the bus-show cycle. In a burst-show cycle only the first data beat is shown externally.

When AT3 = 0 for an access from the core, it indicates either program trace (for an instruction cycle) or reservation (for a data cycle). These indications can also be monitored on two separate signals ($\overline{\text{PTR}}$ and $\overline{\text{RSV}}$), if desired.

- $\overline{\text{PTR}}$ is low when the following is true:
  - AT0 = 0 (Core access)
  - AT2 = 0 (Instruction)

— AT3 = 0 (Program Trace)

- $\overline{\text{RSV}}$ is low when the following is true:
  — AT0 = 0 (Core access)
  — AT2 = 1 (Data)
  — AT3 = 0 (Reservation)

13.4.7.3.5 Burst Data in Progress ($\overline{\text{BDIP}}$)

The master asserts $\overline{\text{BDIP}}$ to indicate to the slave that another data beat follows the current data beat.

13.4.8 Termination Signals

The following sections discuss the termination signals supported by the MPC855T.

13.4.8.1 Transfer Acknowledge ($\overline{\text{TA}}$)

$\overline{\text{TA}}$ indicates normal completion of the bus transfer. The slave asserts $\overline{\text{TA}}$ with every data beat returned or accepted during a burst cycle.

13.4.8.2 Burst Inhibit ($\overline{\text{BI}}$)

The slave asserts $\overline{\text{BI}}$ to indicate to the master that it cannot burst. If this signal is asserted, the master must transfer in multiple cycles and increment the address for the slave to complete the burst transfer.

13.4.8.3 Transfer Error Acknowledge ($\overline{\text{TEA}}$)

Terminates the bus cycle under a bus error condition for which the current cycle is aborted. $\overline{\text{TEA}}$ overrides other cycle termination signals, such as $\overline{\text{TA}}$.

Note that for burst transactions, $\overline{\text{TEA}}$ should be asserted externally only on the first or last beats. Assertion of $\overline{\text{TEA}}$ on an intermediate beat may result in erratic operation, including lockup of the MPC855T requiring hard reset.

13.4.8.4 Termination Signals Protocol

The transfer protocol was defined to avoid electrical contention on lines 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 termination signals should be connected to power through a pull-up resistor to prevent a master from

sampling undefined values in any of these signals when no real slave is addressed. See Figure 13-25 and Figure 13-26.

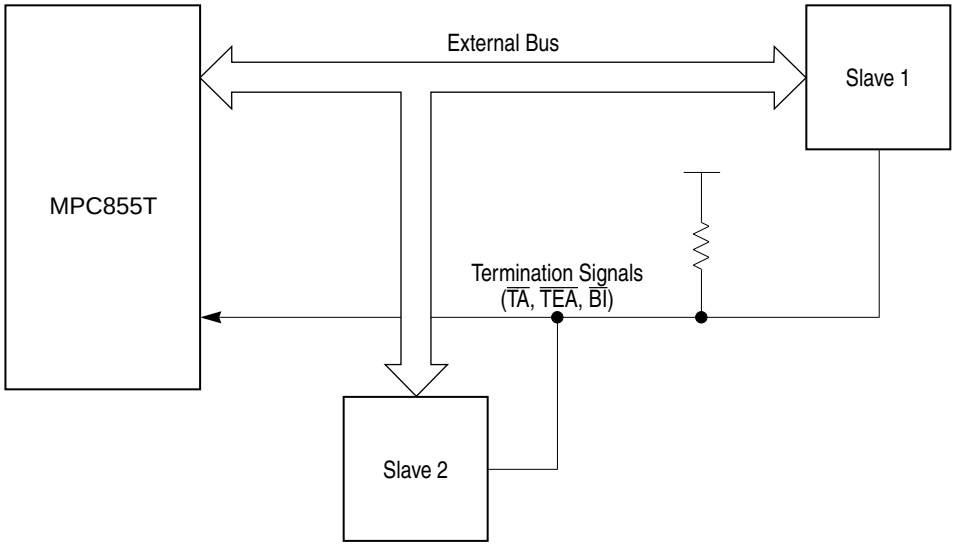


Figure 13-25. Termination Signals Protocol Basic Connection

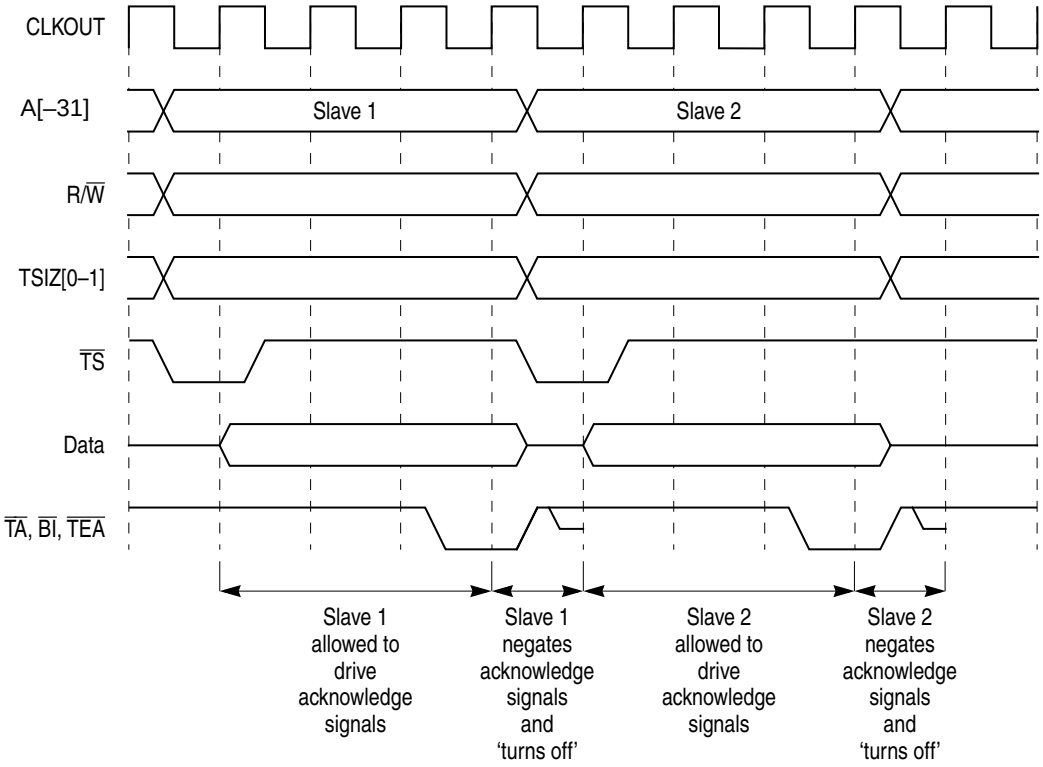


Figure 13-26. Termination Signals Protocol Timing Diagram

13.4.9 Memory Reservation

The MPC855T memory reservation protocol supports multilevel bus structures. For each local bus, reservations are handled by the local reservation logic. The protocol tries to optimize reservation cancellation such that an MPC8xx core processor is notified of memory reservation loss on a remote bus only when it has issued a STWCX cycle to that address. That is, the reservation loss indication comes as part of the STWCX cycle, which avoids the need for fast memory reservation loss indication signals between each remote bus and each MPC8xx master. The memory reservation protocol assumes the following:

- Each processor has no more than 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.
- 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 memory reservation is lost, it is guaranteed that **stwcx.** will not modify the memory.

13.4.9.1 Cancel Reservation ($\overline{CR}$)

$\overline{CR}$ is a point-to-point signal. To use it, reservation logic must remember specifically which bus master requested reservation for which address. If another master writes to the reserved address, the reservation logic asserts $\overline{CR}$ only to the master that holds the associated reservation, thus clearing its flag.

The advantage of $\overline{CR}$ is that it preempts the **stwcx.** instruction if reservation is lost, thus eliminating unnecessary traffic on the external bus.

Figure 13-27 shows the reservation protocol for a single-level (local) bus. It assumes that an external logic on the bus handles the following:

- Snoops accesses to all local bus slaves.
- Holds one reservation for each local master capable of memory reservations.
- Sets the reservation when that master issues a load and reserve request.
- Clears the reservation when another master issues a store to the reservation address.

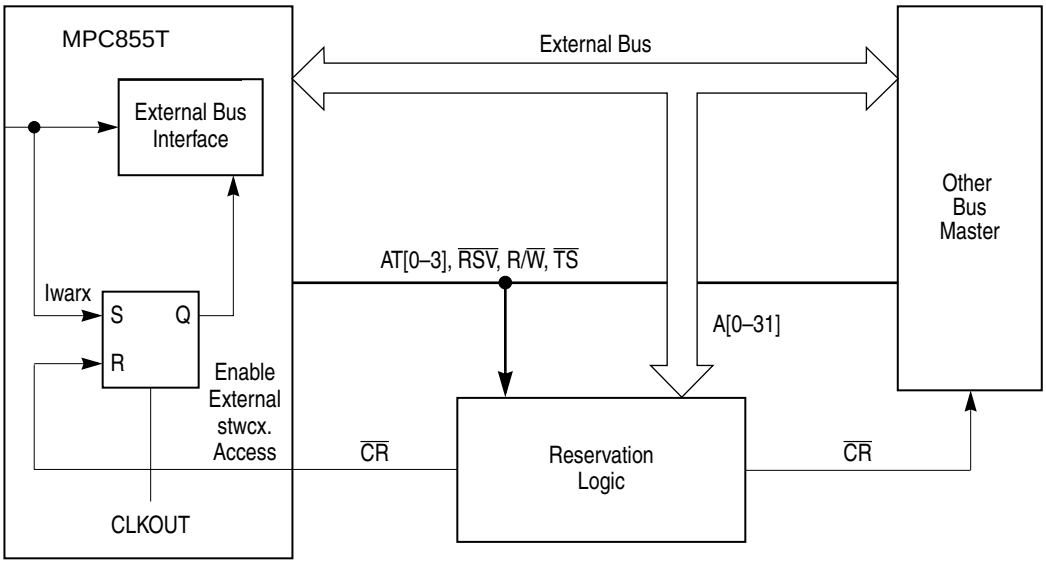


Figure 13-27. Reservation On Local Bus

The MPC855T samples $\overline{CR}$ at the rising edge of CLKOUT. When $\overline{CR}$ is asserted, the reservation flag is reset. The external bus interface samples the logical value of the reservation flag before externally starting a bus cycle initiated by a **stwcx**. instruction in the core. If the reservation flag is set, the external bus interface begins the bus cycle and if it is reset, no bus cycle is initiated externally and this situation is reported to the core.

13.4.9.2 Kill Reservation ($\overline{KR}$)

$\overline{KR}$ is a bused signal. In order to use it, the reservation logic must only remember that one of the bus masters has a reservation for a particular address. If another bus master writes to the address with an instruction other than **stwcx.**, the reservation logic remembers that the reservation for that address was lost. When the master with the reservation subsequently attempts an **stwcx.** instruction to that address, the reservation logic responds to that external bus cycle with $\overline{KR}$.

Note that for burst transactions, $\overline{KR}$ should be asserted externally only on the first or last beats. Assertion of $\overline{KR}$ on an intermediate beat may result in erratic operation, including lockup of the MPC855T requiring hard reset.

Figure 13-28 shows the reservation protocol for a multi-level (local) bus. The system describes a situation in which the reserved location is in the remote bus.

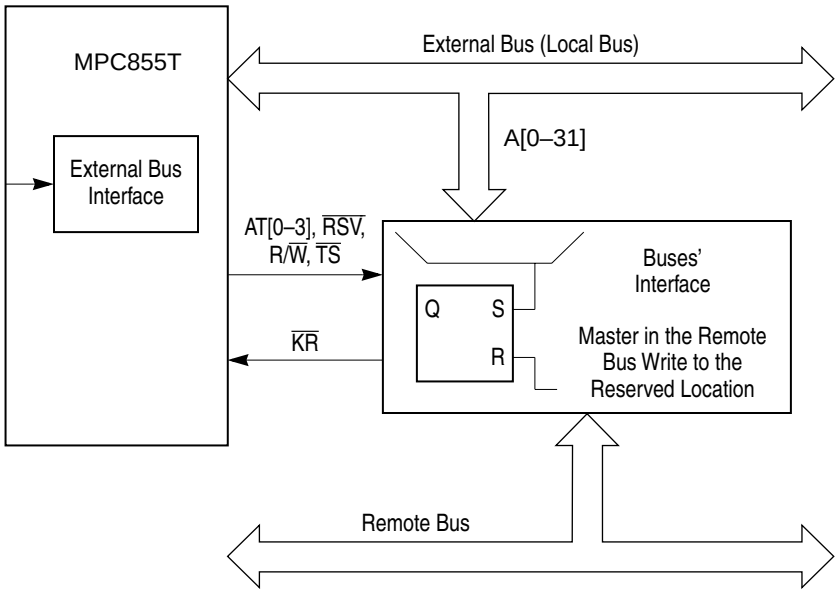


Figure 13-28. Reservation on Multilevel Bus Hierarchy

In this case, the buses' interface block implements a reservation flag for the local bus master. The reservation flag is set by the buses' interface when a load with the local bus master issues a reservation whose address is 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 MPC855T begins a memory cycle to the previously reserved address (located in the remote bus) as a result of a **stwcx**., the following two cases 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 **stwcx.** is reported to the core.

13.4.10 Bus Exception Control Cycles

The MPC855T bus architecture requires assertion of the $\overline{TA}$ from an external device to signal that the bus cycle is complete. $\overline{TA}$ is not asserted in the following cases:

- The external device does not respond
- Various other application-dependent errors occur

External circuitry or the internal MPC855T bus monitor can provide $\overline{TEA}$ when no device responds by asserting $\overline{TA}$ within an appropriate period of time after the MPC855T initiates the bus cycle. This allows the cycle to terminate and the processor to enter exception processing for the error condition (each one of the internal masters causes an internal interrupt under this situation).

To properly control termination of a bus cycle for a bus error, $\overline{\text{TEA}}$ must be asserted at the same time or before $\overline{\text{TA}}$ is asserted. Once $\overline{\text{TEA}}$ is sampled as asserted, it should be negated before the next rising edge to avoid influencing the next initiated bus cycle. $\overline{\text{TEA}}$ is an open-drain pin that allows the wire-OR of different sources of error generation.

13.4.10.1 $\overline{\text{RETRY}}$

When an external device asserts $\overline{\text{RETRY}}$ during a bus cycle, the MPC855T enters a sequence in which it terminates the current transaction, relinquishes bus ownership, and retries the cycle using the same address, address attributes, and data (in the case of a write cycle). Figure 13-29 shows that when the internal arbiter is enabled, the MPC855T negates $\overline{\text{BB}}$ and asserts $\overline{\text{BG}}$ in the clock cycle after $\overline{\text{RETRY}}$ is detected to allow any external master to gain bus ownership. Normal arbitration resumes in the next clock cycle. If the external master does not use the bus, the MPC855T initiates a new transfer with the same address and attributes as before. In Figure 13-30 the same situation is shown where the MPC855T is working with an external arbiter. In this case, in the clock cycle after $\overline{\text{RETRY}}$ is detected asserted, $\overline{\text{BR}}$ and $\overline{\text{BB}}$ are negated together. Normal arbitration resumes one clock cycle later.

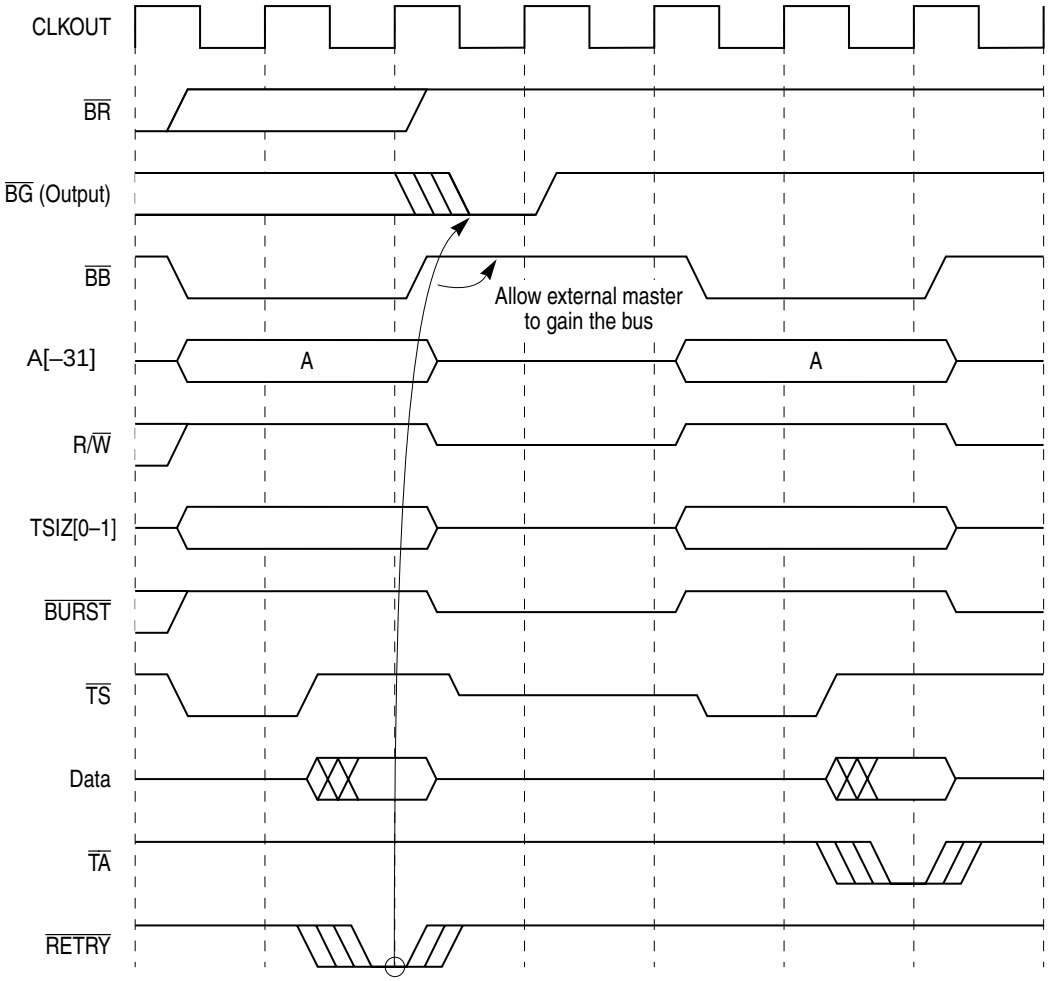


Figure 13-29. Retry Transfer Timing–Internal Arbiter

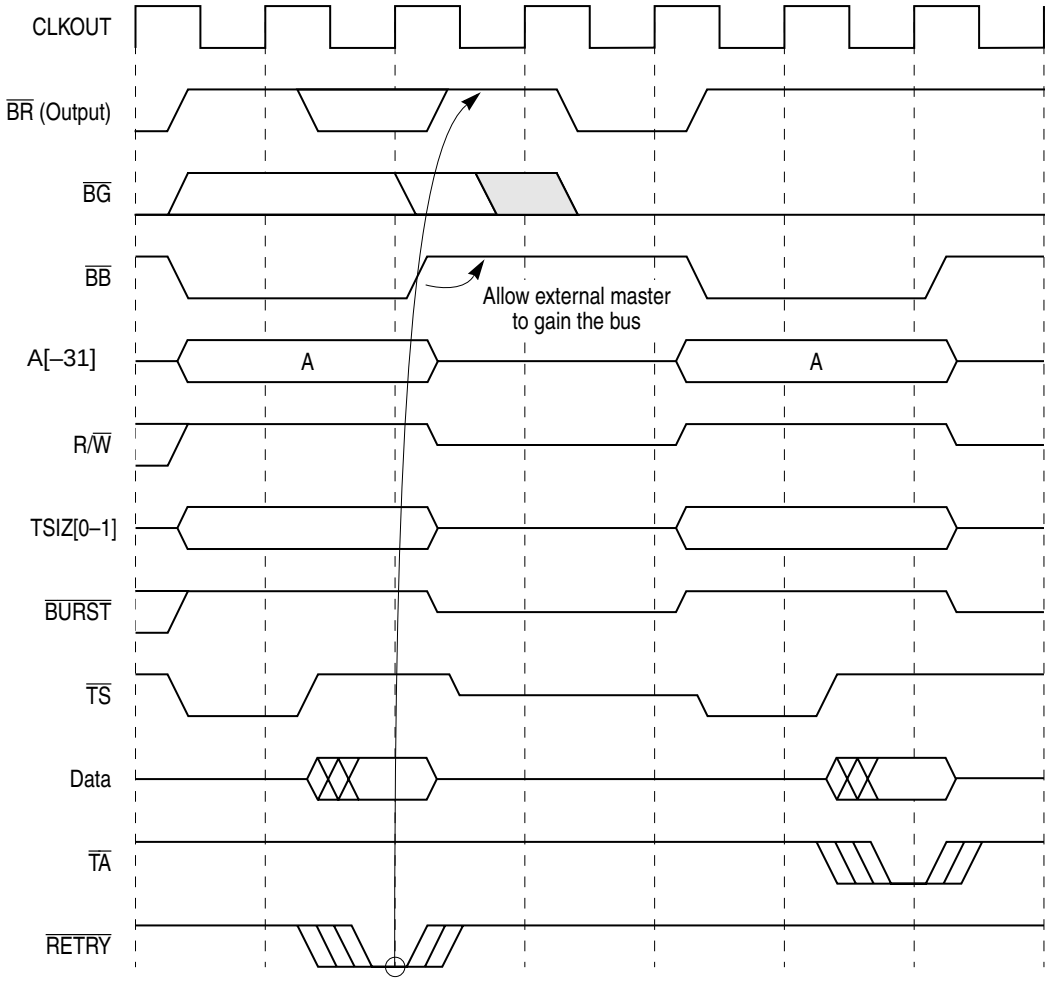


Figure 13-30. Retry Transfer Timing–External Arbiter

When the MPC855T initiates a burst access, the bus interface only recognizes the RETRY assertion as a retry termination if it detects it before the slave device acknowledges the first data beat. Note that for burst transactions, RETRY should be asserted externally only on the first or last beats. Assertion of RETRY on an intermediate beat may result in erratic operation, including lockup of the MPC855T requiring hard reset.

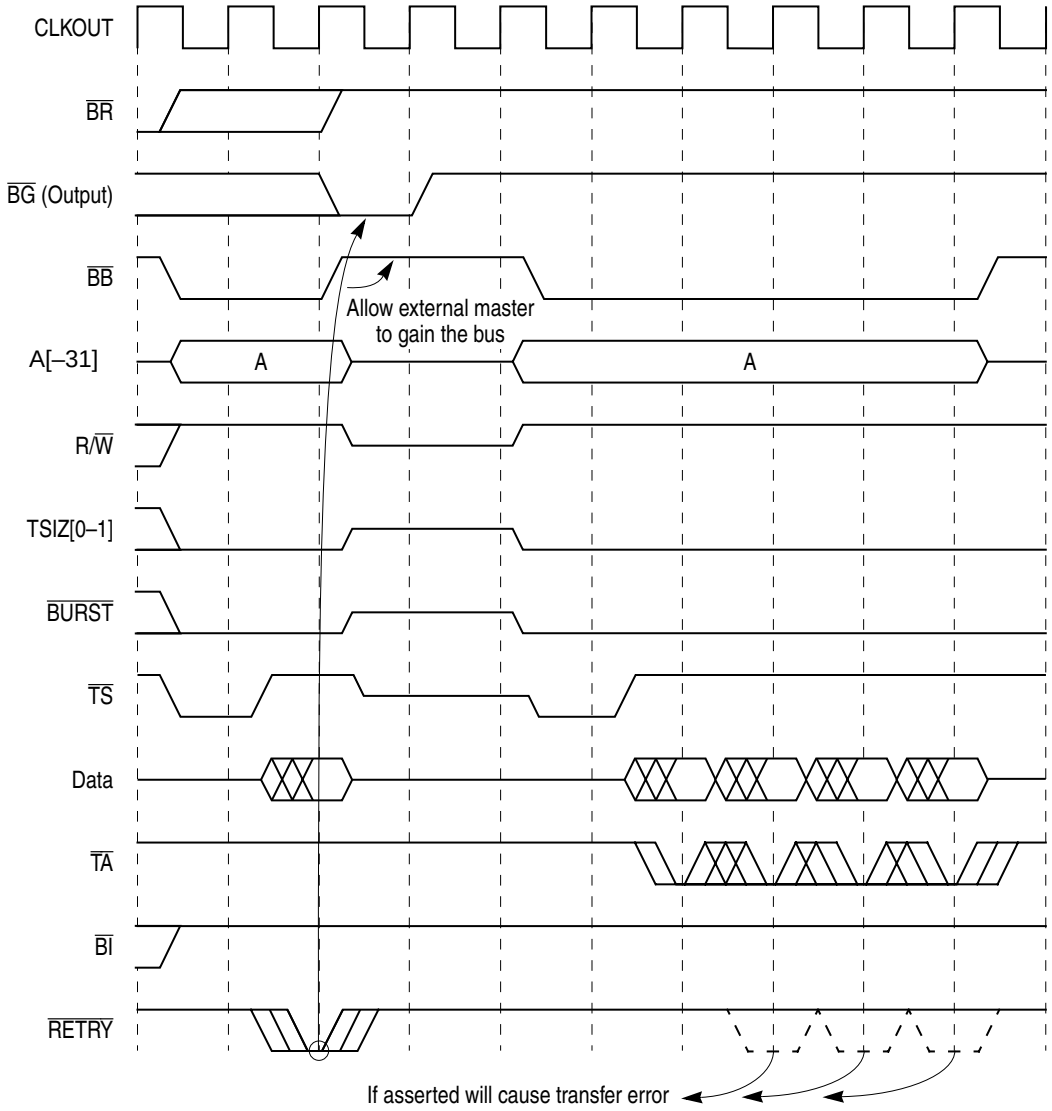


Figure 13-31. Retry on Burst Cycle

If a burst access is acknowledged on its first beat with a normal $\overline{TA}$, but with $\overline{BI}$ asserted, the following single-beat transfers initiated by the MPC855T to complete the 16 byte transfers process the $\overline{RETRY}$ signal assertion as a $\overline{TEA}$. If the MPC855T initiates non-burst access to a small port size device, the transfer size of the access is bigger than the slave port size, and the first transfer of this access is terminated normally by the assertion of $\overline{TA}$, then subsequent single-beat transfers initiated by the MPC855T to complete the access process the $\overline{RETRY}$ assertion as a $\overline{TEA}$.

Table 13-6 summarizes how the MPC855T recognizes the termination signals provided by the slave device that is addressed by the initiated transfer.

Table 13-6. Termination Signals Protocol

$\overline{\text{TEA}}$	$\overline{\text{TA}}$	$\overline{\text{RETRY/KR}}$	Action
0	x	x	Transfer error termination
1	0	x	Normal transfer termination
1	1	0	Retry transfer termination/kill reservation

Chapter 14
Clocks and Power Control

The MPC855T clock system provides many different clocking options for all on-chip and external devices. For its clock sources, the MPC855T contains phase-locked loop and crystal oscillator support circuitry. The phase-locked loop circuitry can be used to provide a high-frequency system clock from a low-frequency external source. Also, to enable flexible power control, the MPC855T provides frequency dividers and a variety of low-power mode options.

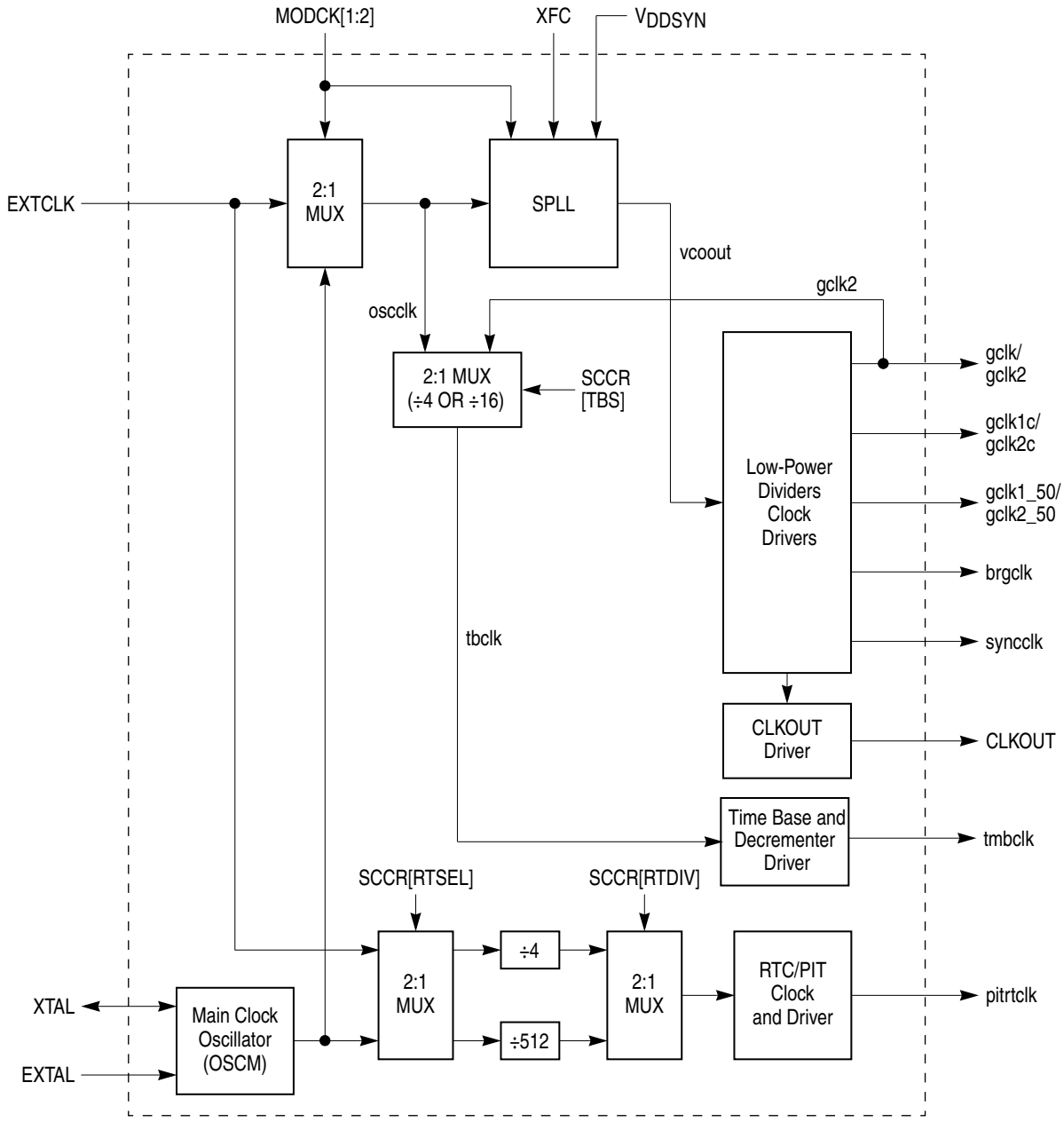
The MPC855T allows a system to optimize power utilization by providing performance on-demand. This is implemented through a variety of programmable power-saving modes with automatic wake-up features.

Figure 14-1 illustrates internal clock source and distribution that includes the system phase-locked loop (SPLL), clock dividers, drivers, and crystal oscillator.

14.1 Features

The main features of the MPC855T clocks and power control system are as follows:

- Contains system PLL (SPLL)
- Supports crystal oscillator circuits
- Clock dividers are provided for low-power modes and internal clocks
- Contains five major power-saving modes
  - Normal (high and low)
  - Doze (high and low)
  - Sleep
  - Deep sleep
  - Power down



Note that only CLKOUT is an actual external output; all other outputs are internal signals.

Figure 14-1. Clock Source and Distribution

14.2 The Clock Module

The clock module consists of two external reference sources and a programmable phase-locked loop, arranged as shown in Figure 14-2.

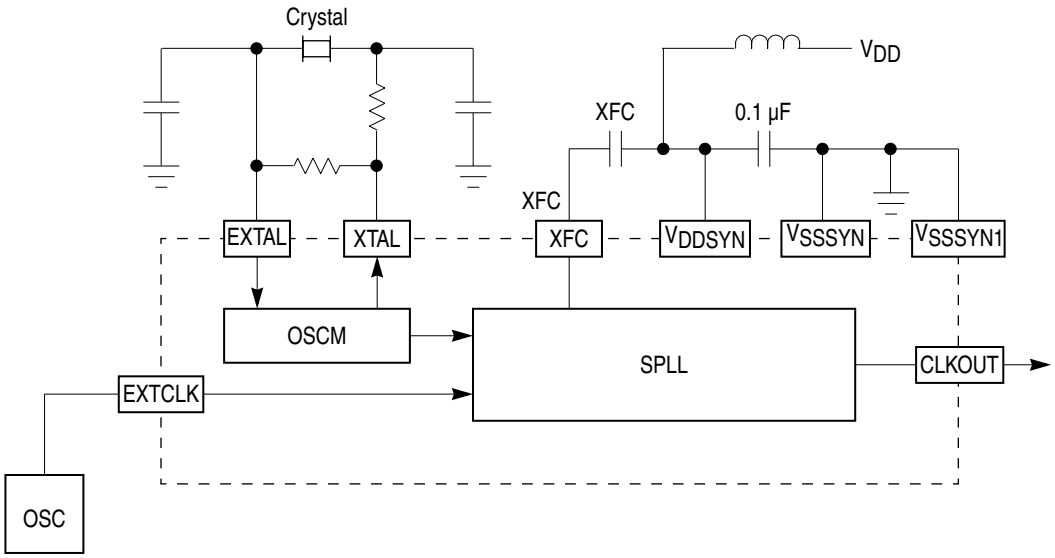


Figure 14-2. Clock Module Components

14.2.1 External Reference Clocks

The MPC855T has two input clock sources, provided at the EXTCLK pin or at the EXTAL and XTAL pins. These two clock sources can select to drive three internal clock signals, referred to as OSCCLK, PITRTCLK, and TMBCLK. OSCCLK provides the input clock to the phase-locked loop. PITRTCLK and TMBCLK provide dedicated clocks for special system timer circuitry, which includes the periodic interrupt timer (PIT), real-time clock (RTC), timebase (TB), and decrementer (DEC) in the SIU. These separate clock sources for the PIT, RTC, TB, and DEC are provided to enable these modules to continue to count at a fixed, user-defined rate regardless of system frequency.

The clock sources for OSCCLK, PITRTCLK, and TMBCLK are selected at reset. The sources for PITRTCLK and TMBCLK can also be selected in software by manipulation of SCCR; see Section 14.6.1, “System Clock and Reset Control Register (SCCR).” For more information, see Section 14.2.2.1, “SPLL Reset Configuration”, Section 14.3.2, “The PIT and RTC Clock (PITRTCLK)”, and Section 14.3.3, “The Time Base and Decrementer Clock (TMBCLK).”

It is possible to use both clock sources in a system, with each providing reference for different functions. If either reference source is not used, then its input should be grounded.

It is not recommended to select the crystal oscillator circuit as OSCCLK while also driving a high-frequency clock source on EXTCLK. This is because noise from the EXTCLK clock source will couple into the crystal oscillator circuit, and will in many cases not allow the system phase-locked loop (SPLL) to lock. The converse, however, is allowable; EXTCLK can be selected as OSCCLK, while the crystal oscillator circuit supplies a separate low-frequency reference.

A typical configuration uses a canned oscillator (4 MHz or 50 MHz) with the EXTCLK input selected as OSCCLK, and a 32.768 kHz or 38.4 kHz crystal at EXTAL and XTAL to provide PITRTCLK.

14.2.1.1 Off-Chip Oscillator Input (EXTCLK)

The external clock input EXTCLK is generated from an external source, which is typically a canned oscillator. The acceptable frequency range of this input source is defined by:

1. The maximum operating frequency of the MPC855T
2. The default SPLL multiplying factor (defined in Section 14.2.2.1, “SPLL Reset Configuration”)
3. The minimum operating frequency of the SPLL, which is 15 MHz

14.2.1.2 Crystal Oscillator Support (EXTAL and XTAL)

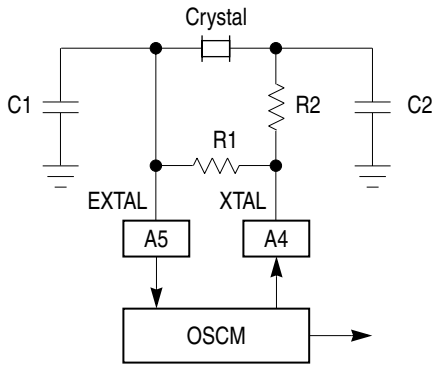
The MPC855T provides support for crystal oscillator circuits with the oscillator module (OSCM). The OSCM has two different modes, supporting two different ranges of frequencies: 30–50 kHz (referred to as 32 kHz mode) or 3-5 MHz (referred to as 4 MHz mode). The mode of OSCM is selected simultaneously with SPLL configuration; refer to Section 14.2.2.1, “SPLL Reset Configuration.

The clock source of OSCM can be provided by a crystal circuit or an external oscillator. If an external oscillator is used, it should be connected to EXTAL, and XTAL should be left unconnected. If a crystal circuit is used, it should be connected between EXTAL and XTAL. The crystal circuit is composed of an on-chip inverting amplifier, an external parallel resonant crystal, two capacitors, and two resistors, as shown in Figure 14-3. EXTAL is the amplifier input for the crystal circuit; XTAL is the amplifier output.

Example values for the passive components of the crystal circuits are provided in Figure 14-3. However, because this is a sensitive analog circuit, these values cannot be guaranteed. These components may have to be tuned due to design-specific parasitic capacitance variation due, for example, to layout and board composition. Careful consideration must be given to component placement and layout, keeping components as near as possible to the chip and keeping all trace lengths to a minimum. It should be noted that the sensitivity of crystal circuits to external component values is so great that even probing the circuit will change its behavior to the point that it may fail to resonate. In practice, experimentation will be required to find an acceptable range of component values, with the final design value being selected in the middle of this range.

Lastly, it should also be noted that future changes in the device technology (shrinks) may change the characteristics of the input and output impedance of the on-chip amplifier. Motorola reserves the right to perform these changes, and designers should be prepared to modify their crystal circuits appropriately should these changes cause their crystal circuit designs to fail. This risk should be taken into account when the design is performed; if

potential manufacturing downtime due to redesign of crystal circuits is unacceptable, a canned oscillator circuit should be used instead.



32 kHz: R1=20MΩ, R2=330kΩ, C1=20pF, C2=20pF

4 MHz: R1=10MΩ, R2=1kΩ, C1=47 pF, C2=56 pF

Figure 14-3. Crystal Circuit Examples

14.2.2 System PLL

The programmable phase-locked loop, called the system phase-locked loop (SPLL) in the MPC855T, generates the overall system operating frequency in integer multiples of the input clock frequency. The SPLL reference clock (OSCCLK) can be generated from either of the external clock sources described in Section 14.2.1, “External Reference Clocks.”

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, providing two advantages. First, lower frequency clock input reduces the overall electromagnetic interference generated by the system. Second, the programmability of the oscillator enables the system to operate at a variety of frequencies with only a single external clock source. The MPC855T SPLL block diagram is shown in Figure 14-4.

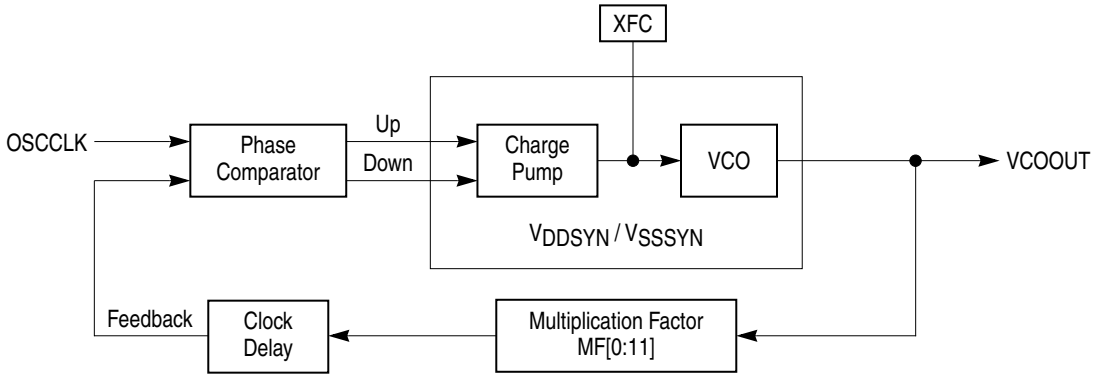


Figure 14-4. SPLL Block Diagram

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 a voltage-controlled oscillator (VCO). The VCO output frequency (VCOOUT) is divided down and fed back to the phase comparator to be compared with the reference frequency (OSCCLK signal). The multiplication factor is programmable in the PLPRCR[MF] between 1 and 4,096.

The minimum VCOOUT operating frequency of the SPLL is 15 MHz. This condition must be maintained both by the reset configuration settings of the SPLL and at the final operating frequency of the SPLL.

The OSCCLK can be supplied by either a crystal or an external clock oscillator. Crystals are typically much cheaper than clock oscillators; however, a clock oscillator has significant design advantages over a crystal circuit in that clock oscillators are easier to work with, resulting in faster design, debugging and production.

Furthermore, it should be noted that low-frequency crystals should not be used for the source of OSCCLK if high-frequency SPLL operation is desired. This is because the default startup multiplication factor of the SPLL requires a loop filter capacitor (XFC) which is incompatible with the capacitor value required at the final operating frequency. For example, if a 50 MHz final value was desired and a 32.768 kHz crystal was used, the XFC range allowable by the default SPLL multiplication factor of 513 is 0.27 µF < XFC < 0.47 µF, whereas the final SPLL multiplication factor of 1526 would require an XFC range of 0.79 µF < XFC < 1.40 µF.

14.2.2.1 SPLL Reset Configuration

While $\overline{\text{PORESET}}$ is asserted, the reset configuration of the SPLL is sampled on the MODCK[1-2] pins. The SPLL immediately begins to use the multiplication factor PLPRCR[MF] value and external clock source for OSCCLK determined by the sampled MODCK[1-2] pin and attempts to achieve lock; therefore, the MODCK[1-2] signals should

be maintained steadily throughout $\overline{\text{PORESET}}$ assertion. The MF field is set as shown in Table 14-1. After $\overline{\text{PORESET}}$ is deasserted, the MODCK[1-2] values are internally latched, and the signals applied to MODCK[1-2] can then be changed.

Table 14-1. Power-On Reset SPLL Configuration

MODCK [1–2]	Default MF+1 at Power-On Reset	SPLL Options Selected
00	513	OSCCLK (SPLL input) is $OSCM_{freq}$ [referred to as 32 kHz mode]
01	5	OSCCLK (SPLL input) is $OSCM_{freq}$ [referred to as 4 MHz mode]
10	1	OSCCLK (SPLL input) is $EXTCLK_{freq}$
11	5	OSCCLK (SPLL input) is $EXTCLK_{freq}$

Note that under no condition should the voltage on MODCK1 and MODCK2 exceed the power supply voltage VDDH applied to the part.

At power-on reset, before the PLL achieves lock, no internal or external clocks are generated by the MPC855T, which may cause higher than normal static current during the short period of stabilization.

NOTE

Upon assertion of HRESET, CLKOUT frequency is not guaranteed.

14.2.2.2 SPLL Output Characteristics and Stability

The minimum frequency at which the SPLL is guaranteed to operate is 15 MHz; therefore, the MPC855T must be configured so that at all times (both after initial system reset and at the final operating frequency) the minimum frequency of CLKOUT is 15 MHz. The maximum frequency at which the SPLL is guaranteed to operate is the maximum rated frequency of the part (for example, 50 MHz for a 50-MHz part).

The multiplication factor is the most important parameter in defining the SPLL stability. There are three factors related to the multiplication factor that define SPLL stability:

- Phase skew—The time difference between the rising edges of EXTCLK and CLKOUT for a capacitive load on the CLKOUT pin over the entire process, temperature ranges, and voltage ranges. For input frequencies greater than 15 MHz and (MF+1) ≤ 2, this skew is ±0.9 ns. Otherwise, this skew is not guaranteed. However, for (MF+1) < 10 and input frequencies greater than 10 MHz, the skew is ±2.3 ns.
- Phase jitter—A variation in the skew that occurs between the rising edges of EXCLK and CLKOUT 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 15 MHz and (MF+1) ≤ 2, this jitter

is less than ±0.6ns. Otherwise, this jitter is not guaranteed. However, for (MF+1) < 10 and input frequencies greater than 10 MHz, this jitter is less than ±2ns.

- Frequency jitter—The frequency variation of CLKOUT. For small multiplication factors, that is, (MF+1)<10, this jitter is smaller than 0.5%. For mid-range multiplication factors (10<(MF+1)<500), this jitter is between 0.5% and ~2%. For large multiplication factors ((MF+1)>500), the frequency jitter is 2–3%. The maximum input frequency jitter on EXTAL is 0.5%. If the rate of change of the frequency of EXTAL is slow (that is, it does not jump between the minimum and maximum values in one cycle), the maximum jitter can be 2%.

14.2.2.3 System Phase-Locked Loop Pins (VDDSYN, VSSSYN, VSSSYN1, XFC)

The internal frequency of the MPC855T and the output of the CLKOUT pin depend on the quality of the input clock source and the PLPRCR[MF]. 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. For requirements concerning this power supply, refer to Section 14.4.3, “Clock Synthesizer Power (VDDSYN, VSSSYN, VSSYN1).
- VSSSYN and VSSSYN1—Ground reference pins for the analog SPLL circuitry. For requirements concerning this ground reference, refer to Section 14.4.3, “Clock Synthesizer Power (VDDSYN, VSSSYN, VSSYN1).
- 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, the XFC capacitor must be low leakage, with 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. XFC should be selected so that it satisfies both the range of values required by the MF determined at reset and by the MF value programmed as the final operating value.

Table 14-2. XFC Capacitor Values Based on PLPRCR[MF]

MF Range	Minimum Capacitance	Recommended Capacitance	Maximum Capacitance	Unit
1 ≤ (MF+1) ≤ 4	XFC = [(MF+1) x 580] - 100	XFC = [(MF+1) x 680] - 120	XFC = [(MF+1) x 780] - 140	pF
(MF+1) > 4	XFC = (MF+1) x 830	XFC = (MF+1) X 1100	XFC = (MF+1) x 1470	pF

  - Note that the these ranges are not strict cutoffs; they merely represent ranges where the best jitter performance will be achieved. If there is no overlap between two ranges of operation, choose the minimum or maximum value of the

recommended XFC range for the normal operating frequency of the system, whichever is nearest the range for the other frequency.

14.2.2.4 Disabling the SPLL

For special purposes, such as testing, it is possible to disable the SPLL. The SPLL is disabled if VDDSYN is grounded. In this case, VCOOUT will be equal to OSCCLK/2.

Note that because the skew elimination provided by the SPLL is also disabled, this mode of operation invalidates the timing of the MPC855T. Thus, this mode must not be used as a normal operating mode; its only valid use is for low-frequency testing of board integrity during production.

14.3 Clock Signals

The MPC855T uses the following clocks, summarized in Table 14-3. These clocks are described in the following three sections, grouped by their different sources.

Table 14-3. Functionality Summary of the Clocks

Clock	Description
GCLK1C/GCLK2C	Basic clocks supplied to the core, the data and instruction caches, and MMUs.
GCLK1/GCLK2	Basic clocks supplied to the SIU, clock module, CP, and most other features in the CPM
GCLK1_50/GCLK2_50	Optionally divided versions of GCLK1/GCLK2, which are used to clock the GPCM and UPM in the memory controller and to provide the CLKOUT output for the external bus.
BRGCLK	Clocks the four baud rate generators and the memory controller refresh timer. This allows the serial ports to operate at a fixed frequency and the memory refresh to continue at a uniform rate even when the rest of the MPC855T is operating at a reduced frequency (for example, when in normal low or doze low modes)
SYNCCLK	Used by the serial synchronization circuitry in the serial ports of the CPM, and includes the SI, SCC and SMCs. SYNCCLK performs the function of synchronizing externally generated clocks before they are used internally. SYNCCLK allows the SI, SCC, and SMCs to continue operating at a fixed frequency, even when the rest of the MPC855T is operating at a reduced frequency.
CLKOUT	Clock out is an external clock signal used to drive other devices, and thus provide the ability to operate synchronously with those devices. Equivalent to the internal GCLK2_50 signal.
TMBCLK	Clocks the time base and decrementer
PITRTCLK	Clocks the periodic interrupt timer and the real time clock

14.3.1 Clocks Derived from the SPLL Output

The MPC855T uses the following 9 internal clock signals, which are derived from the SPLL output clock (VCOOUT):

- General system clocks—GCLK1C, GCLK2C, GCLK1, GCLK2
- Memory controller and external bus clocks—GCLK1_50, GCLK2_50

- Baud rate generator clock—BRGCLK
- Synchronization clocks—SYNCCLK, SYNCCLKS

The MPC855T also provides the GCLK2_50 signal externally on the CLKOUT pin.

The SPLL output VCOOUT is sent to frequency dividers that generate the GCLKx, GCLKxC, GCLKx_50, SYNCCLK, and BRGCLK which are sent to the rest of the modules of the MPC855T. The division factor for each divider is programmed in the SCCR. The organization of the low-power dividers is shown in Figure 14-5.

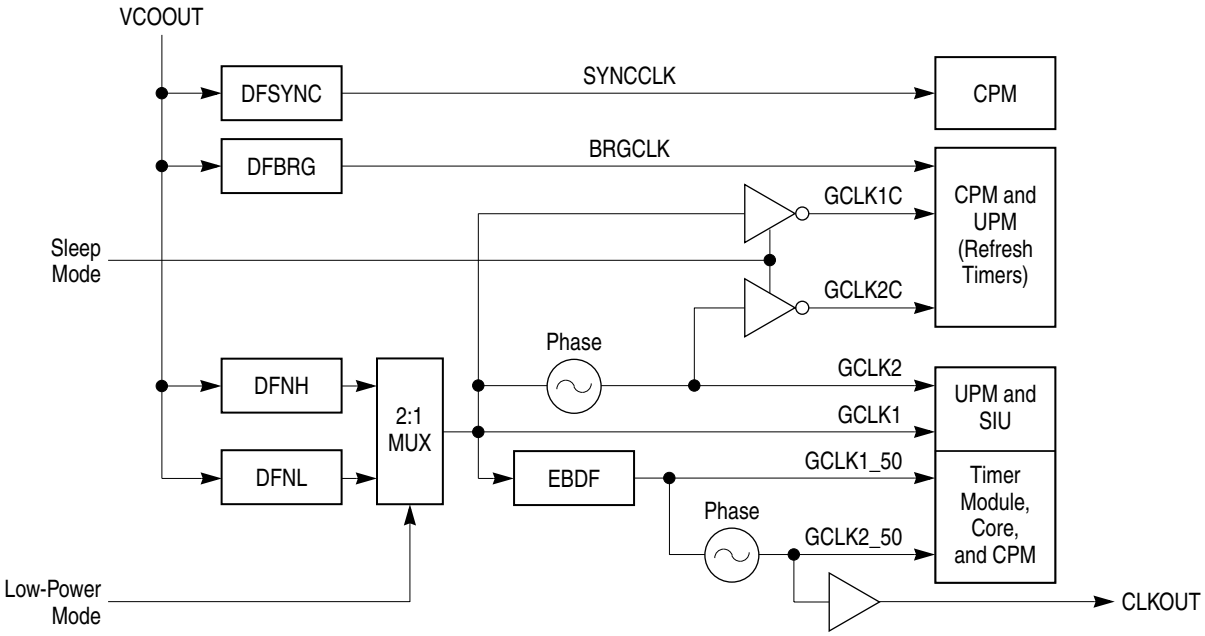


Figure 14-5. Clock Dividers

14.3.1.1 The Internal General System Clocks (GCLK1C, GCLK2C, GCLK1, GCLK2)

The GCLKxC and GCLKx signals are referred to here collectively as GCLKx. The difference between the GCLKxC and GCLKx signals are as follows:

- The GCLKxC clocks are supplied to the core, data and instruction caches, and memory management unit. They are not active when the MPC855T is in doze, sleep, or deep-sleep modes.
- The GCLKx clocks are supplied to the SIU, clock module, memory controller, and most of the other blocks in the CPM. They are not active when the MPC855T is in sleep or deep-sleep modes.

GCLKx can be dynamically switched between two different frequencies determined by dividers programmed in SCCR[DFNH] and SCCR[DFNL], as shown in Figure 14-6.

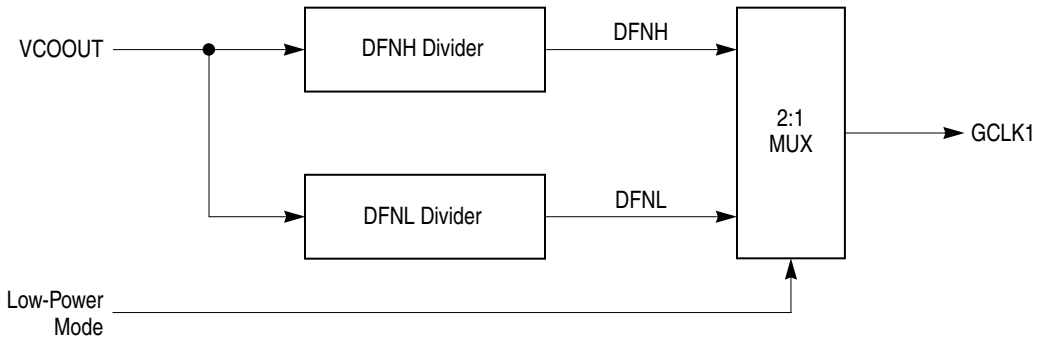


Figure 14-6. Low-power dividers for GCLKx

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. The DFNH and DFNL dividers are cleared by $\overline{\text{HRESET}}$, and therefore GCLKx defaults to VCOOUT.

The frequency for the GCLKx system clock is:

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

When GCLKx is divided, its duty-cycle is modified. One phase remains the same while the other stretches out. GCLKx no longer has a 50% duty cycle when the division factor is greater than 1, as shown in Figure 14-7.

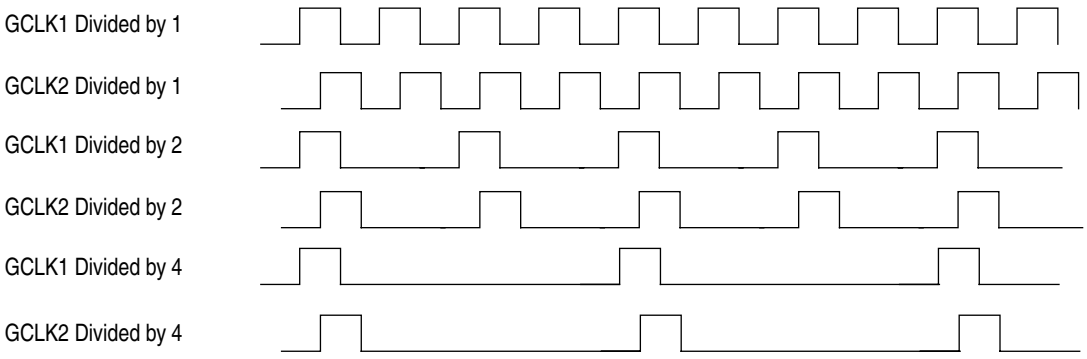


Figure 14-7. Divided System Clocks (GCLKx) Timing Diagram

14.3.1.2 Memory Controller and External Bus Clocks (GCLK1_50, GCLK2_50, CLKOUT)

The MPC8 provides the capability to run the external bus and memory controller at a lower frequency than the internal modules. This capability is provided by the external bus frequency dividers. The external bus clocks GCLK1_50 and GCLK2_50 are derived from GCLK1 and GCLK2, as determined by the SCCR[EBDF]. SCCR[EBDF] is cleared by

$\overline{\text{HRESET}}$, and thus GCLK1_50 and GCLK2_50 default to GCLK1 and GCLK2. The timing relationship between GCLKx and GCLKx_50 is shown in Figure 14-8.

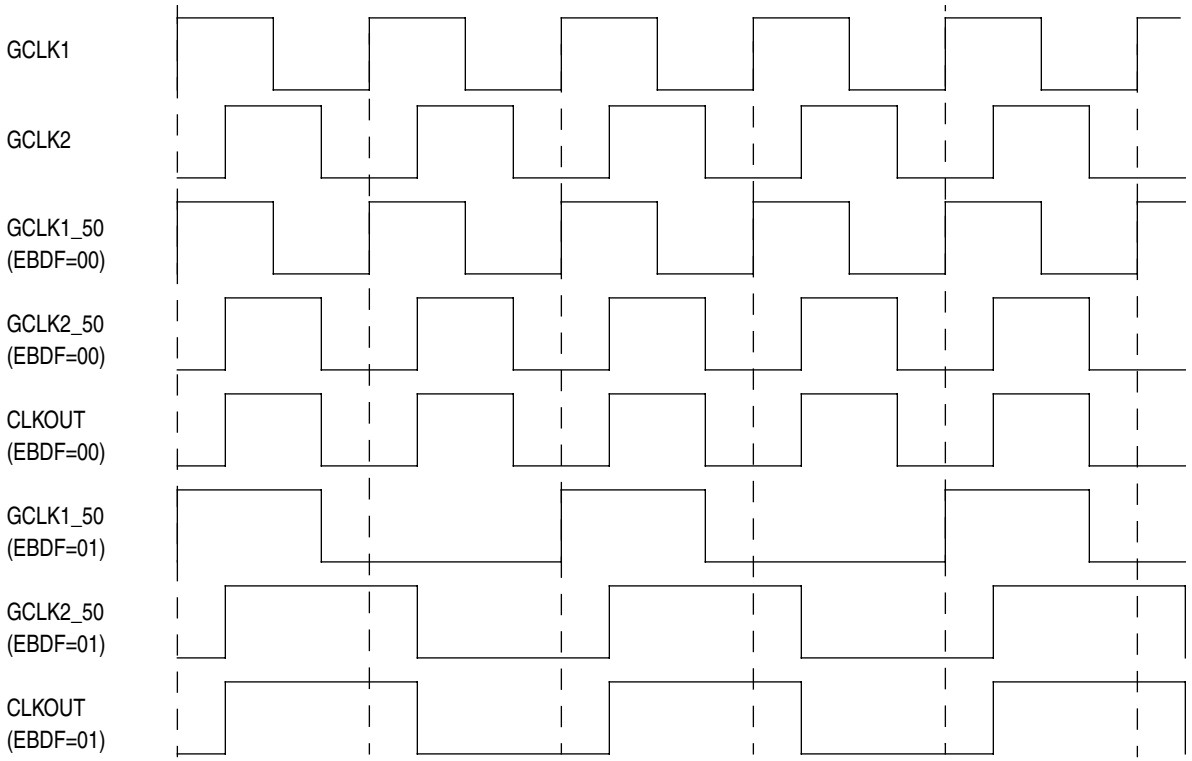


Figure 14-8. Memory Controller and External Bus Clocks Timing Diagram for EBDF=0 and EBDF=1

If SCCR[EBDF]=0, the duty cycle of both GCLK1_50 and GCLK2_50 is 50%. However, if SCCR[EBDF]=1, the duty cycle of GCLK2_50 is 50%, but the duty cycle of GCLK1_50 is 37.5%, as shown in Figure 14-8.

The low-power frequency dividers described in Section 14.3.1.1, “The Internal General System Clocks (GCLK1C, GCLK2C, GCLK1, GCLK2)” also effect the frequency and duty cycle of GCLK1_50, GCLK2_50, and CLKOUT. For an example of this, see Figure 14-9.

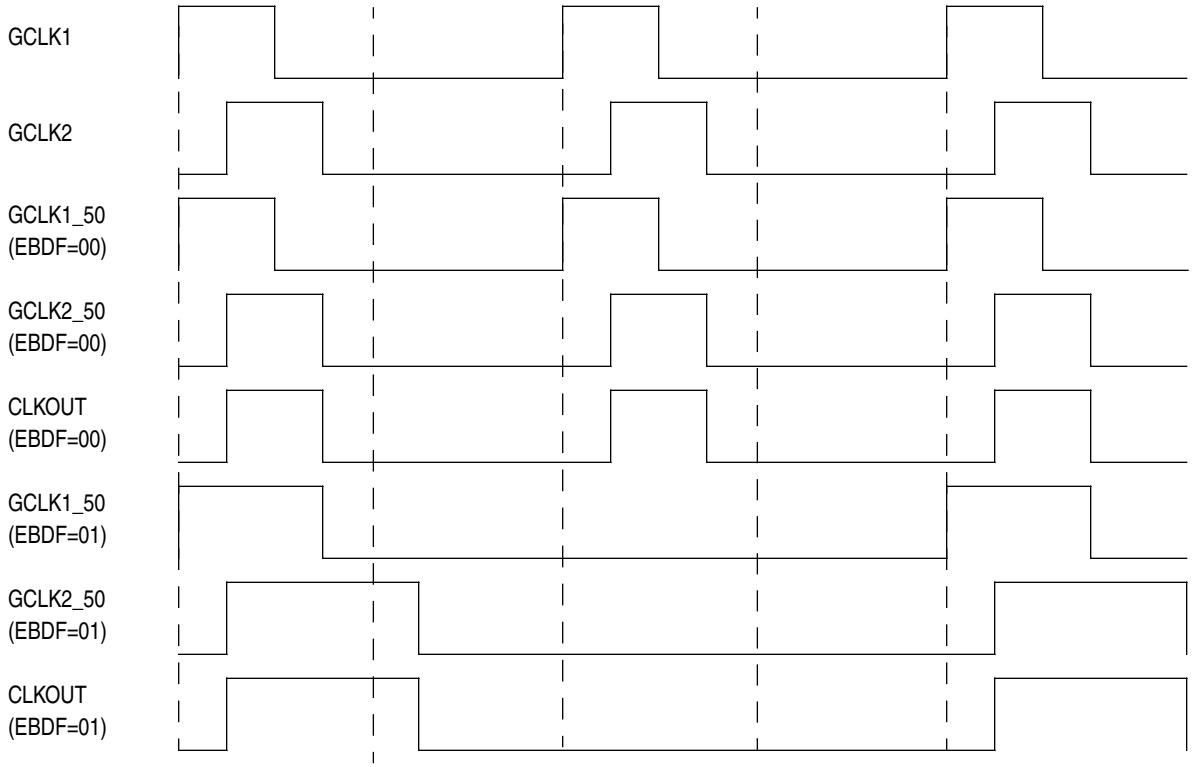


Figure 14-9. Memory Controller and External Bus Clocks Timing Diagram for (CSRC=0 and DFNH=1) or (CSRC=1 and DFNL=0)

The frequency of GCLK1_50 and GCLK2_50 are affected both by the SCCR[DFNH] and SCCR[DFNL] dividers and by the SCCR[EBDF] divider. Thus, the frequency for GCLKx_50 and CLKOUT is as follows:

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

CLKOUT is the only externally visible clock, and is equivalent to the internal signal GCLK2_50. 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. While the SPLL is acquiring lock, the CLKOUT signal does not oscillate and remains in a low state.

14.3.1.3 CLKOUT Special Considerations: 1:2:1 Mode

To enable synchronization of a system to the EXTCLK signal while still allowing the internal circuits of the MPC855T to operate at an increased frequency, it is necessary to maintain synchronization of the EXTCLK and CLKOUT signal. Specifically, this operation entails:

- input clock source EXTCLK
- internal clock of 2xEXTCLK, provided by multiplying EXTCLK by 2 in the SPLL (by programming PLPRCR[MF]=1)
- external bus clock CLKOUT with frequency equivalent to EXTCLK, provided by dividing GCLK2 by 2 (by programming SCCR[EBDF]=01)

This is also known as 1:2:1 mode. In this mode, in order to allow multiple devices clocked by the same EXTCLK source to maintain synchronization on the external bus, EXTCLK and CLKOUT must be in phase. This operation cannot be guaranteed on MPC855Ts prior to revision . On MPC855Ts of revision or later, this operation can be guaranteed, but it requires that SCCR[EBDF] be written first, followed by the write to PLPRCR[MF].

14.3.1.4 The Baud Rate Generator Clock (BRGCLK)

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. The baud rate generator clock is controlled independently in order to allow the baud rate generators and memory refresh rate to continue operating at a fixed frequency, even when the rest of the MPC855T is operating at a reduced frequency.

BRGCLK defaults to VCOOUT, but can be reduced in frequency by a frequency divider. This frequency divider is controlled by SCCR[DFBRG].

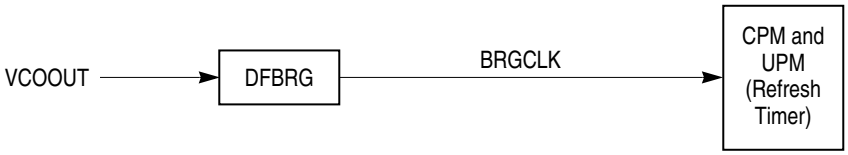


Figure 14-10. BRGCLK Divider

The baud rate generator clock frequency is as follows:

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

14.3.1.5 The Synchronization Clock (SYNCCLK, SYNCCLKS)

The synchronization clock signals (SYNCCLK and SYNCCLKS, referred to collectively as SYNCCLK) are used by the signal synchronization circuitry in the serial ports of the

communication processor module. The signal synchronization circuitry is used to sample and synchronize asynchronous external signals provided to these ports. 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 MPC855T is operating at a reduced frequency.

SYNCCLK defaults to VCOOUT, but can be reduced in frequency by a frequency divider. This frequency divider is controlled by SCCR[DFSYNC].

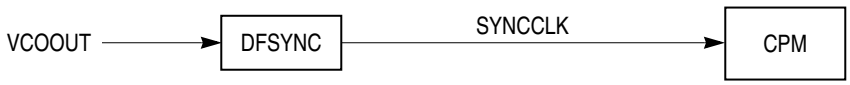


Figure 14-11. SYNCCLK Divider

The synchronization clock frequency is as follows:

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

Limitations on SYNCCLK include the following:

- SYNCCLK must always have a frequency at least as high as GCLKx.
- SYNCCLK must and be at least two times the maximum serial clock rate used by the serial ports in the system.
- If the time-slot assigner (TSA) is used, SYNCCLK must be at least 2.5 times the maximum serial clock rate of the TSA.

14.3.2 The PIT and RTC Clock (PITRTCLK)

The PIT and RTC clock is generated either from EXTCLK or the crystal oscillator circuit (OSCM). This input source can be divided by either 4 or 512. The PITRTCLK source and divide factor are selected by SCCR[RTSEL] and SCCR[RTDIV].

When used by the real-time clock (RTC), the PITRTCLK source is first divided as determined by RTDIV, and then divided in the RTC circuits by either 8192 or 9600. Therefore, in order for the RTC to count in seconds, the clock source must satisfy:

(EXTCLK or OSCM) / [(4 or 512) x (8192 or 9600)] = 1

The RTC will operate with other frequencies, but it will not count in units of seconds.

If there were only one clock source for the system, this requirement would limit the set of desirable frequencies at which to operate the MPC855T. However, the MPC855T provides two independent clock sources, EXTCLK and OSCM. To allow for maximum flexibility in system frequency selection independent of real-time clock operation, it is recommended

that a 32.768 kHz or 38.4 kHz crystal with the OSCM be used for the PITRTCLK source if the RTC is to be used.

The MODCK[1-2] state at $\overline{\text{PORESET}}$ deassertion determines the input clock source and prescaler value for PITRTCLK. These values can be changed after reset by manipulating the associated bits in the SCCR.

Table 14-4. PITRTCLK Configuration at $\overline{\text{PORESET}}$

MODCK [1:2]	PITRTCLK Prescaler SCCR[RTDIV]	PITRTCLK Input Source SCCR[RTSEL]
00	4	OSCM (crystal oscillator)
01	512	OSCM (crystal oscillator)
10	512	EXTCLK
11	512	EXTCLK

14.3.3 The Time Base and Decrementer Clock (TMBCLK)

The time base and decrementer clock is generated either from the input frequency of the SPLL (OSCCLK) or the general system clock GCLK2. The SCCR[TBS] bit is used to select between these two sources.

The MODCK[1-2] state at $\overline{\text{PORESET}}$ deassertion, the SCCR[TBS], and the SPLL multiplication factor determine the input clock source and prescaler value for TMBCLK.

Table 14-5. TMBCLK Configuration

SCCR[TBS]	MODCK[1-2] at $\overline{\text{PORESET}}$	MF + 1	Clock Source	TMBCLK Prescaler
1	XX	X	GCLK2	16
0	0X	X	OSCCLK	4
0	1X	1, 2	OSCCLK	16
0	1X	> 2	OSCCLK	4

14.4 Power Distribution

The various modules of the MPC855T are connected to four distinct power rails. These power rails have different requirements, as explained in the following sections. The organization of the power rails is shown in Figure 14-12.

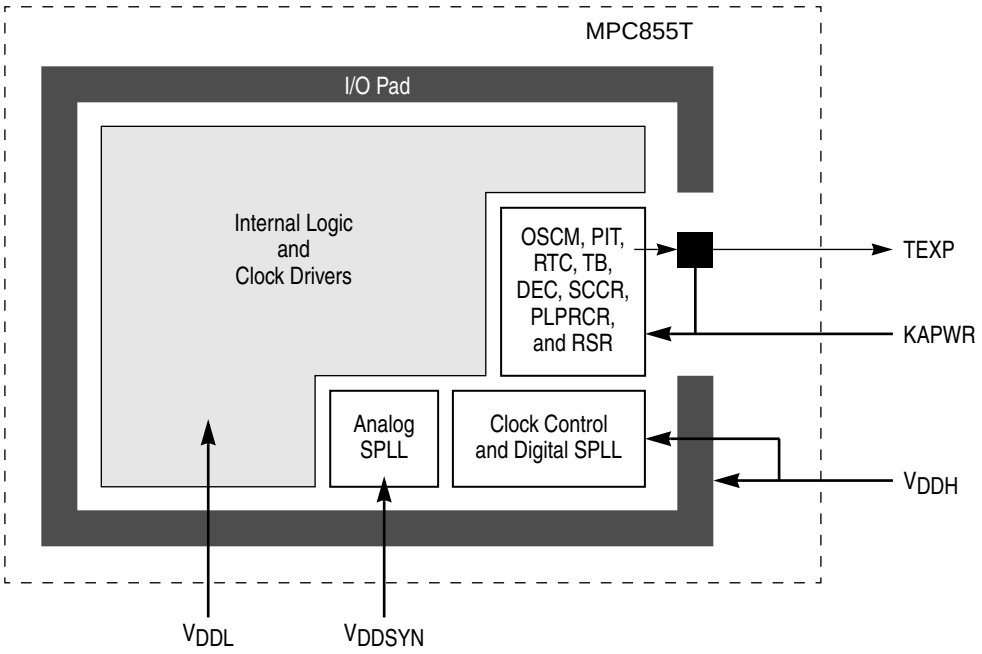


Figure 14-12. MPC855T Power Rails

A complete tabulation of modules and power supplies is given in Table 14-6.

Table 14-6. MPC855T Modules vs. Power Rails

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

14.4.1 I/O Buffer Power (VDDH)

The I/O buffers, logic, and clock control are fed by a 3.3V power supply.

VDDH must in all cases be greater than or equal to VDDL.

14.4.2 Internal Logic Power (VDDL)

The internal logic can be fed by the same 3.3V source which powers VDDH. VDDL is identified as a separate power supply only to facilitate power measurements.

14.4.3 Clock Synthesizer Power (VDDSYN, VSSSYN, VSSYN1)

To improve stability, the power supply pins for the SPLL are uniquely identified in order to allow special filtration to be provided for them.

A well-regulated voltage should be applied to VDDSYN via a low impedance path to the VDDH/VDDL power rail. The allowable noise on the VDDSYN power plane is 20 mV peak up to a bandwidth of 100 MHz. This typically requires isolation of the VDDSYN power plane from the VDDH/VDDL power plane. An example implementation of this is a split power plane, with the VDDSYN plane implemented as an island in the VDDH/VDDL power plane, connected to the VDDH/VDDL power plane with an inductor and to the ground plane with bypass capacitors. An inductor value of 8.2 mH and bypass capacitor values of 0.1 µF and 10 µF provide a two-pole filter with a cutoff frequency of 500 Hz. Note that this example noise filter implementation is taken from a Motorola MPC860 ADS board and may need to be modified for a user's particular system design, board and power supply to meet the VDDSYN noise requirement.

VSSSYN and VSSSYN1 must have a low impedance path to the ground plane. If sufficient isolation is provided for VDDSYN (as described above), no additional isolation for VSSSYN and VSSSYN1 is required.

14.4.4 Keep-Alive Power (KAPWR)

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 the system to remove the power at the VDDH/VDDL/VDDSYN pins during power-down mode.

14.5 Power Control (Low-Power Modes)

To optimize power consumption, the MPC855T provides low-power modes that can be used to dynamically activate and deactivate certain internal modules, such that only the needed modules are operating at any given time. In addition to normal high mode (i.e. fully activated), the MPC855T supports normal low, doze high, doze low, sleep, deep-sleep, and power-down modes.

In addition to these power-saving modes, it should be noted that the architecture of the CPM inherently supports optimum power consumption. When the CPM is idle, it uses its own power-saving mechanism to shut down automatically.

Low-power modes are controlled in the PLPRCR[LPM] and PLPRCR[CSRC]. Events can cause automatic changes from one low-power mode to another. These events include software-initiation (through the MSR[POW]), CPM activity, internal interrupt sources, external interrupt sources, and resets. These events are enabled in the SCCR[PRQEN].

The characteristics of each low-power mode are summarized in Table 14-7. Table 14-7 also provides equations for approximate power consumption equations for each of these modes.

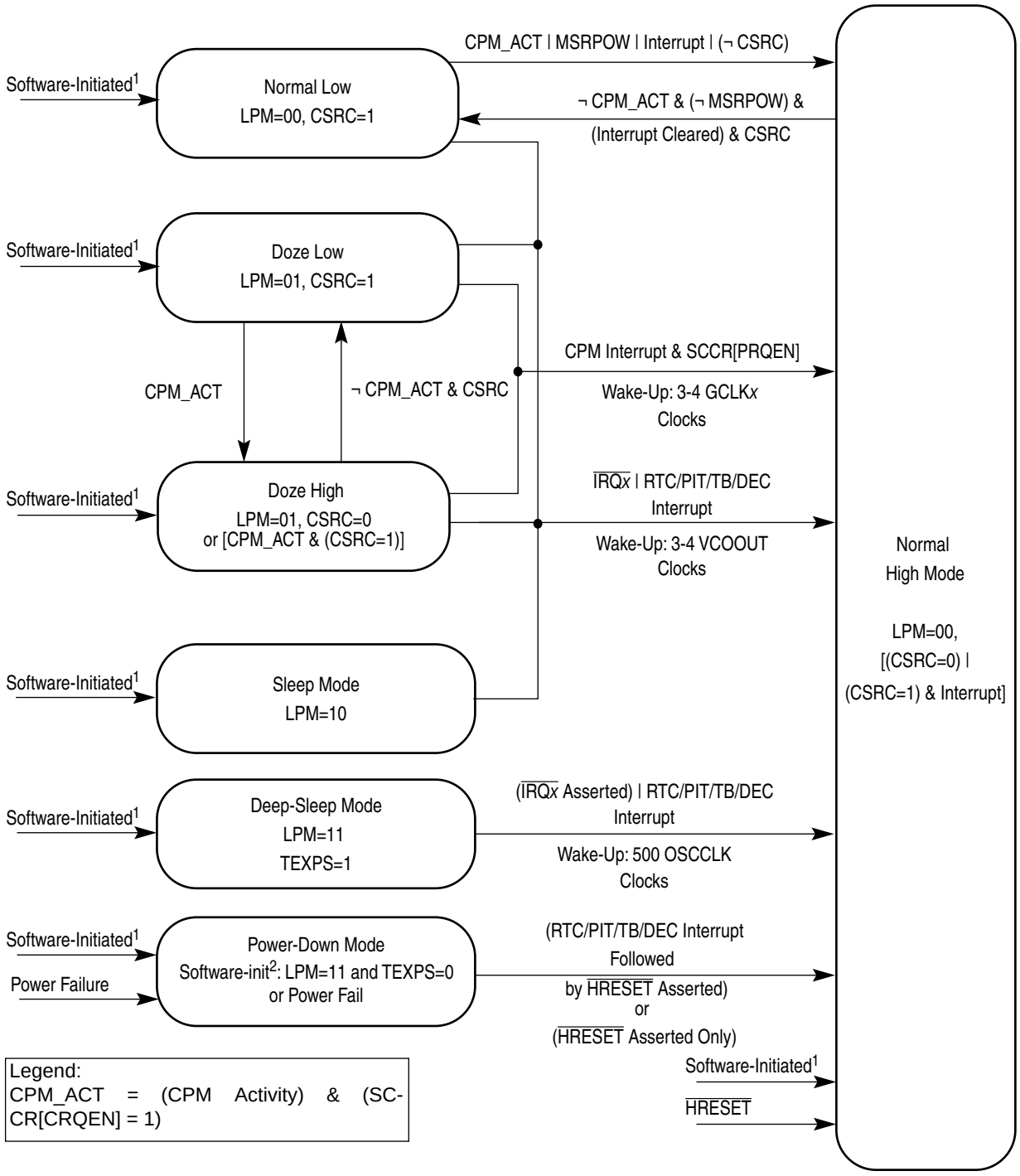
Table 14-7. MPC855T Low-Power Modes

Operation Mode	SPLL	GCLK*x* Frequency	Wake-Up Method	Return Time from Wake-Up Event to Normal High	Typical MPC855T Power Consumption at 50 MHz	Functionality
Normal high LPM=00	Active	VCOOUT $\div 2^{DFNH}$	—	—	20 mW + $1/2^{DFNH}$ W ≅	Full
Normal low LPM=00	Active	VCOOUT $\div 2^{DFNL+1}$	Software-Initiation, or Internal or External Interrupt	Asynchronous exceptions: 3-4 VCOOUT Clocks Synchronous exceptions 3-4 GCLK2 Clocks	20 mW + $1/2^{(DFNL+1)}$ W ≅	
Doze high LPM=01	Active	VCOOUT $\div 2^{DFNH}$	Internal or External Interrupt		20 mW + $0.4/2^{DFNH}$ W ≅	Enabled: SIU timers, CPM, and memory controller Disabled: core, MMU, caches
Doze low LPM=01	Active	VCOOUT $\div 2^{DFNL+1}$	Internal or External Interrupt		20 mW + $0.4/2^{(DFNL+1)}$W ≅	

Table 14-7. MPC855T Low-Power Modes (continued)

Operation Mode	SPLL	GCLK*x* Frequency	Wake-Up Method	Return Time from Wake-Up Event to Normal High	Typical MPC855T Power Consumption at 50 MHz	Functionality
Sleep LPM=10	Active	Inactive	Interrupt from RTC, PIT, DEC, TB, IRQx	3-4 VCOOUT Clocks	<10 mW	Enabled: RTC, periodic interrupt timer, timebase, and decrementer
Deep-sleep LPM=11 TEXPS=1	Inactive	Inactive	Interrupt from RTC, PIT, DEC, TB, IRQx	<500 OSCM Clocks 16ms-32 kHz	TBD	
Power-down LPM=11 TEXPS=0	Inactive	Inactive	Interrupt from RTC, PIT, DEC, TB followed by external hard reset	<500 OSCM clocks + power supply wake-up (PwSp_Wake+ 16 ms at 32 kHz)	32 kHz ~35µA, KAPWR = 2.7V (if applied directly), 3.0V (if going through a diode to KAPWR pin, recommended) Temperature = 50˚ C	

A state diagram describing transitions between the various low-power modes is shown in Figure 14-13.



1 Software is active only in normal high/low modes.

2 Software initiation of power-down mode requires that the TEXP output be used by external logic to gate main power (VDDH, VDDL, and VDDSYN).

Figure 14-13. MPC855T Low-Power Mode Flowchart

14.5.1 Normal High Mode

Normal high mode is the default mode of the MPC855T. In this mode, the GCLKx frequency is determined by SCCR[DFNH], and all modules of the MPC855T are enabled. For more information about SCCR[DFNH], refer to Section 14.3.1.1, “The Internal General System Clocks (GCLK1C, GCLK2C, GCLK1, GCLK2).

Normal high mode is selected if PLPRCR[CSRC]=0 and PLPRCR[LPM]=00, or if an enabled event has caused an exit from another low-power mode.

14.5.2 Normal Low Mode

Normal low mode takes advantages of the low-power dividers for GCLKx to enable full functionality of the MPC855T, but at a lower frequency so that power consumption is reduced. The low-power dividers allow the system to reduce and restore the operating frequencies of different sections of the MPC855T without losing the SPLL lock. This mode is sometimes referred to as slow-go or low gear mode.

Normal low mode is selected if PLPRCR[CSRC]=1 and PLPRCR[LPM]=00. In normal low mode, the GCLKx frequency is determined by SCCR[DFNL]. For more information about SCCR[DFNL], see Section 14.3.1.1, “The Internal General System Clocks (GCLK1C, GCLK2C, GCLK1, GCLK2).” Note also that PLPRCR[TMIST] should be cleared before entering normal low mode; for more information, see Section 14.5.8, “TMIST: Facilitating Nesting of SIU Timer Interrupts.”

Normal low mode can be entered at any time, and the frequency of operation of normal low mode can be changed dynamically. This is controlled by PLPRCR[CSRC] and SCCR[DFNL]. Changes to these bits take effect immediately.

The following events cause the MPC855T to leave normal low mode and enter normal high mode:

- A pending interrupt from the interrupt controller occurs. This option is maskable with SCCR[PRQEN]. These interrupts include all internal and external interrupt sources, if enabled.
- Software-initiation, by writing MSR[POW] = 0. This option is maskable with SCCR[PRQEN].
- The communications processor (CP) has a service request from a peripheral (SCC, SMC, etc.). This option is maskable with SCCR[CRQEN].

14.5.3 Doze High Mode

When software initiates the doze high mode, software processing on the core suspends. The GCLKxC clocks to the core, MMUs, and caches are disabled. However, the CPM and SIU continue to function as normal.

Doze high mode is selected if PLPRCR[CSRC]=0, MSR[POW]=1, and PLPRCR[LPM]=01. In doze high mode, the GCLKx frequency is determined by SCCR[DFNH]. For more information about SCCR[DFNH], see Section 14.3.1.1, “The Internal General System Clocks (GCLK1C, GCLK2C, GCLK1, GCLK2).” Note also that PLPRCR[TMIST] should be cleared before entering doze high mode; for more information, see Section 14.5.8, “TMIST: Facilitating Nesting of SIU Timer Interrupts.”

The MPC855T leaves doze high mode and enter normal high mode when a pending interrupt from the interrupt controller occurs. These interrupts include all internal and external interrupt sources, if enabled. This action requires that SCCR[PRQEN] be set; otherwise, the MPC855Twill not wake up. When the MPC855T enters normal high mode, PLPRCR[LPM] is cleared.

Upon resumption of processing in normal high or low mode, the MPC855T jumps to the external interrupt vector to process the interrupt source. When the core returns from the exception handler via **rfi**, it resumes processing from the instruction following that which initiated entry into doze mode.The one exception to this is the decrementer, a wake-up interrupt from the decrementer never causes a jump to the interrupt handler; instead processing always resumes from the instruction following that which initiated entry into low-power mode.

14.5.4 Doze Low Mode

Doze low mode is similar to Doze high mode, except that additionally the system clock frequency has been reduced. In doze low mode, the GCLKx frequency is determined by SCCR[DFNL]. For more information about SCCR[DFNL], see Section 14.3.1.1, “The Internal General System Clocks (GCLK1C, GCLK2C, GCLK1, GCLK2).”

Doze low mode is selected if PLPRCR[CSRC]=1, MSR[POW]=1, and PLPRCR[LPM]=01. Note also that PLPRCR[TMIST] should be cleared before entering doze low mode; for more information, see Section 14.5.8, “TMIST: Facilitating Nesting of SIU Timer Interrupts.”

The MPC855T has the option to temporarily leave doze low mode and enter doze high mode when CPM activity occurs. This option is enabled in SCCR[CRQEN]. When the CP finishes servicing the peripheral request, the MPC855T automatically reenters doze low mode.

The MPC855T leaves doze low mode and enter normal high mode when a pending interrupt from the interrupt controller occurs. These interrupts include all internal and external interrupt sources, if enabled. This action requires that SCCR[PRQEN] be set; otherwise, the MPC855T will not wake up. When the MPC855Tenters normal high or normal low mode, PLPRCR[LPM] is cleared.

When the MPC855T leaves doze low mode, it enters normal high mode if SCCR[PRQEN] is set; otherwise it enters normal low mode.

Upon resumption of processing in normal high or low mode, the MPC855T jumps to the external interrupt vector to process the interrupt source. When the core returns from the exception handler via **rfi**, it resumes processing from the instruction following that which initiated entry into doze mode. The one exception to this is the decrementer, a wake-up interrupt from the decrementer never causes a jump to the interrupt handler; instead processing always resumes from the instruction following that which initiated entry into low-power mode.

14.5.5 Sleep Mode

In sleep mode, the only internal modules that are activated are the SIU timers, including the real-time clock (RTC), periodic interrupt timer (PIT), timebase (TB), and decrementer (DEC).

Sleep mode is selected if PLPRCR[LPM]=10. Only PITRTCLK and TMBCLK are active in sleep mode. Clocks to all other modules are disabled. Note that because the SIU memory controller is not activated in this mode, memory refresh does not occur. Note also that PLPRCR[TMIST] should be cleared before entering sleep mode; for more information, see Section 14.5.8, “TMIST: Facilitating Nesting of SIU Timer Interrupts.”

The following events cause the MPC855T to leave sleep mode and enter normal high mode:

- An external $\overline{\text{IRQx}}$ input is asserted for which wake-up capabilities are enabled. Wake-up capabilities for $\overline{\text{IRQx}}$ interrupts are enabled in the associated SIEL[WMx] bits.
- A time-out event of the RTC, PIT, TB, or DEC occurs.

When the MPC855T leaves sleep mode, it enters normal high or normal low mode, depending on the state of PLPRCR[CSRC] and SCCR[PRQEN]. When the MPC855T enters normal high mode, PLPRCR[LPM] is cleared.

Upon resumption of processing in normal high or low mode, the MPC855T jumps to the external interrupt vector to process the interrupt source if that interrupt is enabled in SIMASK and MSR[EE]. When the core returns from the exception handler via **rfi**, it resumes processing from the instruction following that which initiated entry into sleep mode. The one exception to this is the decrementer, a wake-up interrupt from the decrementer never causes a jump to the interrupt handler; instead processing always resumes from the instruction following that which initiated entry into low-power mode.

14.5.6 Deep-Sleep Mode

Deep-sleep mode is similar to sleep mode, except that the SPLL is also disabled and, therefore, the wake-up time from this mode is longer. Wake-up time from deep-sleep mode is a maximum of 500 OSCCLK clocks (if OSCCLK is sourced by OSCM) or a maximum of 1000 clocks (if OSCCLK is sourced by EXTCLK).

Deep-sleep mode is selected if PLPRCR[LPM]=11 and PLPRCR[TEXPS]=1. Note also that PLPRCR[TMIST] should be cleared before entering deep-sleep mode; for more information, see Section 14.5.8, “TMIST: Facilitating Nesting of SIU Timer Interrupts.”

Note that the RTC, PIT, TB, and DEC operate in deep-sleep mode only if their timing reference is OSCM. In all other aspects, the behavior of deep-sleep mode is identical to that of sleep mode.

14.5.7 Power-Down Mode

Power-down mode describes the condition where a power source is applied to KAPWR, but the power source for VDDH, VDDL, and VDDSYN has been shut down. The behavior in this mode is similar to deep-sleep mode, in that the SPLL is shut down and only the real-time clock (RTC), periodic interrupt timer (PIT), timebase (TB), and decrementer (DEC) are active. The RTC, PIT, TB, and DEC operate in power-down mode only if their timing reference is OSCM.

In normal operation, KAPWR should be greater than or equal to approximately (VDDH - 0.4) V. In power-down mode, KAPWR should be greater than or equal to 2.0 V.

If power-down mode is used, connect KAPWR to both VDDH and the back-up battery through diodes. To prevent battery current from being drawn when VDDH is active, the back-up battery voltage should be greater than 2.4 V but less than (VDDH - 0.4) V. If, however, power-down mode is not used, tie KAPWR directly to VDDH.

Exiting from power-down mode requires a full hardware reset. Note that if it is required that the PIT, TB, DEC, and SPLL registers and settings not change during power-down mode and the subsequent reset, then $\overline{\text{PORESET}}$ should be pulled high throughout power-down mode and $\overline{\text{HRESET}}$ should be used for the reset during wake-up. Otherwise, $\overline{\text{PORESET}}$ can be used for this reset source. After initial power-up, $\overline{\text{PORESET}}$ assertion does not affect the RTC registers.

To maintain stability of the crystal oscillator, switchover between the main power supply and KAPWR supply should be done smoothly. The maximum power supply rise time seen at the KAPWR pin should be less than 1.7 V/ms for a 32-kHz input frequency. This can be done by connecting a capacitor from KAPWR to ground.

Power-down mode can be used for:

- A software-initiated controlled shutdown, with optional automatic wakeup,
- Maintaining integrity of the real-time clock (RTC) during a power failure.

14.5.7.1 Software Initiation of Power-Down Mode, with Automatic Wake-up

Power-down mode can be initiated in software if the external TEXP signal is used to control the power supply for VDDH, VDDL, and VDDSYN. If software clears TEXPS, the TEXP signal deasserts. This signal deassertion can be used externally to shut down the VDDH, VDDL, and VDDSYN power supplies. In performing this operation, TEXP should be deasserted by setting PLPRCR[LPM]=11 and clearing PLPRCR[TEXPS] (by writing 1).

The TEXP signal can also be used to enable automatic or externally-initiated wakeup from power-down mode. When the RTC, PIT, TB, or DEC generate an event, or when $\overline{\text{HRESET}}$ is asserted externally, PLPRCR[TEXPS] is set and the TEXP pin is asserted. TEXP can be externally connected to a switch that turns on the power supply to the chip, as shown in Figure 14-14. The MPC855T should then go through a normal hard reset sequence. When performing this hard reset sequence, it is important to allow enough time for the oscillator to warm up and the SPLL to lock.

In this configuration, the following pins must be connected as listed below, in order to keep them from being unintentionally sampled as asserted and causing an unintended exit from power-down mode.

- EXTCLK and $\overline{\text{RSTCONF}}$ must be connected through a pull-down resistor (10K Ohms) to ground.
- $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ must be connected through a pull-up resistor (100K Ohms) to VDD.
- $\overline{\text{PORESET}}$ must be connected through a pull-up resistor (47K Ohms) to KAPWR.
- KAPWR must be connected to both the main power supply, and the keep-alive power supply through diodes.

This scheme is shown in Figure 14-14.

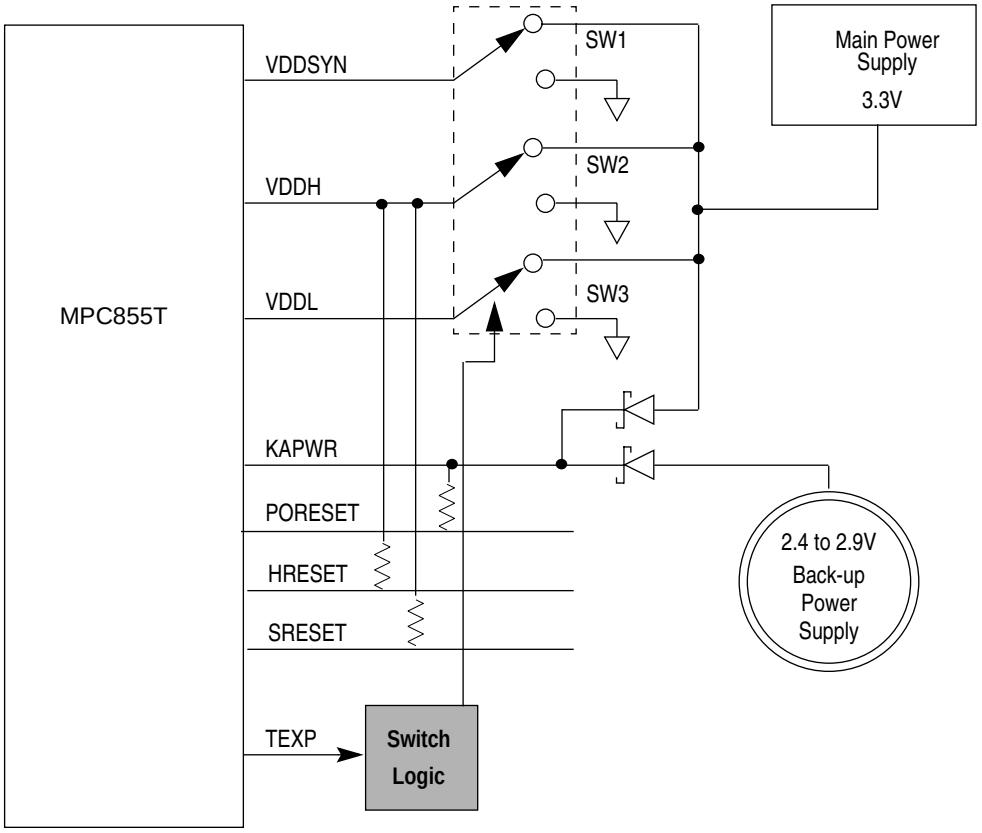


Figure 14-14. Software-Initiated Power-Down Configuration

Switches for VDDH, VDDL, and VDDSYN are shown separately; however, if they are supplied from the same source, there would actually be only a single switch. When PLPRCR[TEXPS] is cleared, TEXP is deasserted and the power is shut down. PLPRCR[TEXPS] is asserted by the MPC855T 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, or when the $\overline{\text{HRESET}}$ signal is externally asserted.

14.5.7.2 Maintaining the Real-Time Clock (RTC) During Shutdown or Power Failure

The power-down configuration can be used simply to maintain integrity of the real-time clock (RTC) if a power shutdown or power failure should occur. The backup KAPWR source is used to maintain the RTC. In this configuration, no provision is made for automatic wake-up from power-down mode. Instead, it is assumed that the appropriate reset sequence will be initiated when the power supply to VDDH, VDDL, and VDDSYN is reapplied.

Some power monitoring circuits drive a reset signal when the power supply voltage falls below a specified threshold, in order to assure that erratic behavior does not occur in a low-voltage situation. This functionality can be used with the MPC855T,while still maintaining integrity of the RTC.

The reset signal from the power monitor circuit should be connected to the $\overline{\text{PORESET}}$ signal of the MPC855T. If power dips below the threshold, $\overline{\text{PORESET}}$ is driven to the MPC855T, which resets all of the modules of the MPC855T except the RTC. If power fails entirely, $\overline{\text{PORESET}}$ remains asserted, but the RTC continues to operate if a backup power supply (battery) is connected to KAPWR.

In this configuration, $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ should be pulled up to VDDH, not to KAPWR. This is because assertion of $\overline{\text{PORESET}}$ causes the MPC855T to assert $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Pulling these signals up to KAPWR causes current to drain unnecessarily. If $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ are pulled up to VDDH and VDDH is not powered, then no current drain will result from the $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ assertion. $\overline{\text{PORESET}}$ should be pulled up to KAPWR. EXTCLK and RSTCONF should both be pulled down to ground.

14.5.7.3 Register Lock Mechanism: Protecting SIU Registers in Power-Down Mode

If the MPC855T sets PLPRCR[LPM]=11 before entering power-down mode, then the registers of the SIU maintained by KAPWR are automatically protected. However, to provide protection of the SIU registers maintained by KAPWR against uncontrolled shutdown, a register locking mechanism is included. These registers can be write-protected in a set of associated key registers. For more information on the register lock mechanism, see Section 10.4.5, “Register Lock Mechanism.”

14.5.8 TMIST: Facilitating Nesting of SIU Timer Interrupts

It is often desirable, within an interrupt service routine, to clear the source of the interrupt at the beginning of the routine, in order to facilitate nesting of interrupts. However, if normal low mode is enabled, clearing an interrupt source can cause transition into normal low mode, which may not be desired. In order to resolve these conflicting interests, PLPRCR[TMIST] is provided. A timeout in the RTC, PIT, TB, or DEC will cause the PLPRCR[TMIST] to be set. While PLPRCR[TMIST] is set, entry into low-power mode is disabled. Thus, the SIU timer interrupt source can be cleared immediately in the interrupt service routine, while still allowing entry into low-power mode to be enabled at a later, user-defined time (when software clears PLPRCR[TMIST]). Note, however, this requires that PLPRCR[TMIST] must be cleared before entry into any low-power mode other than normal high mode.

14.6 Clock and Power Control Registers

The following sections describe the clock and power control registers.

14.6.1 System Clock and Reset Control Register (SCCR)

The SPLL has a 32-bit control register that is powered by keep-alive power. The system clock and reset control register (SCCR), shown in Figure 14-15, is memory-mapped into the MPC855T SIU's register map.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—	COM		—			TBS	RTDIV	RTSEL	CRQEN	PRQEN	—		EBDF		—
HRESET	—	#		0			#	#	#	0	0	0		†		0
POR	0	0		0			0	*	*	0	0	0		†		0
R/W	R/W															
Addr	(IMMR&0XFFFF0000) + 280															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—	DFSYNC		DFBRG		DFNL			DFNH			—			—	
HRESET	0															
POR	0															
R/W	R/W															
Addr	(IMMR&0XFFFF0000) + 282															

Note: HRESET is hard reset and POR is power-on reset.
The field is undefined
— The field is unaffected.
* RTDIV depends on the combination of MODCK1 and MODCK2. RTSEL depends on MODCK1. See Table 14-4 for more information.
† This field is set according to the default of the hard reset configuration word.

Figure 14-15. System Clock and Reset Control Register (SCCR)

This register is affected by $\overline{\text{HRESET}}$ but is not affected by $\overline{\text{SRESET}}$. Table 14-8 describes SCCR fields.

Table 14-8. SCCR Field Descriptions

Bits	Name	Description
0	—	Reserved, should be cleared.
1–2	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, clock output should be disabled 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.
3–5	—	Reserved, should be cleared.
6	TBS	Timebase source. Determines the clock source that drives the timebase and decrementer. 0 Timebase frequency source is the OSCCLK divided by 4 or 16. 1 Timebase frequency source is GCLK2 divided by 16.
7	RTDIV	Real-time clock divide. Determines 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.
8	RTSEL	Real-time clock select. Selects the crystal oscillator or main clock oscillator as the input source to PITRTCLK. At power-on reset, it reflects the value of MODCK1. 0 OSCM (crystal oscillator) is selected. 1 EXTCLK is selected.
9	CRQEN	CPM request enable. Cleared by power-on or hard reset. In low-power modes, specifies if the general system clock returns to high frequency while the CP is active. 0 The system remains in low frequency even if the communication processor module is active. 1 The system switches to high frequency when the communication processor module is active.
10	PRQEN	Power management request enable. In low-power modes, specifies whether the general system clock returns to a high frequency when a pending interrupt from the interrupt controller or MSR[POW] is clear (normal mode). 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 MSR[POW] = 0 (normal mode). 1 The system switches to high frequency when there is a pending interrupt from the Interrupt controller or MSR[POW] = 0.
11–12	—	Reserved, should be cleared.
13–14	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 11.3.1.1, “Hard Reset Configuration Word.” 00 CLKOUT is GCLK2 divided by 1. 01 CLKOUT is GCLK2 divided by 2. 1x Reserved.
15–16	—	Reserved, should be cleared.

Table 14-8. SCCR Field Descriptions (continued)

Bits	Name	Description
17–18	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.
19–20	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.
21–23	DFNL	Division factor low frequency. Sets the VCOOUT frequency division factor for general system clocks to be used in low-power mode. In low-power mode, the MPC855T 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 changing 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.
24–26	DFNH	Division factor high frequency. Sets the VCOOUT frequency division factor for general system clocks to be used in normal mode. In normal mode, the MPC855T automatically switches to the DFNH frequency. To select the DFNH frequency, load this field with the divide value and clear CSRC. A loss-of-lock condition does not occur when this field is changed. 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.
27–31	—	Reserved, should be cleared.

14.6.2 PLL, Low-Power, and Reset Control Register (PLPRCR)

The 32-bit system PLL, low-power, and reset control register (PLPRCR), shown in Figure 14-16, is powered by a keep-alive power supply and is used to control the system frequency and low-power mode operation.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	MF												—			
HRESET	—												0			
POR	*												0			
R/W	R/W															
Addr	(IMMR&0xFFFF0000) + 284															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	SPLSS	TEXPS	—	TMIST	—	CSRC	LPM		CSR	LOLRE	FIOPD	—				
HRESET	—	1	0	0	0	0	0		—	—	—	0				
POR	0	1	0	0	0	0	0		0	0	0	0				
R/W	R/W															
Addr	(IMMR&0xFFFF0000) + 286															

NOTE: HRESET is hard reset and POR is power-on reset.
* Depends on the combination of MODCK1and MODCK2. See Table 14-4 for more information.

Figure 14-16. PLL, Low-Power, and Reset Control Register (PLPRCR)

This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Table 14-9 describes PLPRCR bits.

Table 14-9. PLPRCR Field Descriptions

Bits	Name	Description
0–11	MF	Multiplication factor. Determines the factor by which the OSCCLK input is multiplied to produce VCOOUT. This field controls the value of the divider in the SPLL feedback loop. Programmable between 1and 4096, where 0x000 corresponds to 1 and 0xFFF corresponds to 4096. The MF field can be read and written at any time but do not change MF value unless core is fully serialized ICTRL[ISCT_SER]=000. Changing the MF field causes the SPLL to lose its lock. All clocks are disabled until the SPLL reaches lock condition.
12–15	—	Reserved, should be cleared.
16	SPLSS	System PLL lock status sticky. Cleared by power-on reset. 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). SPLSS is affected only by an unintentional loss of lock due to a hardware-related issue. A software-initiated loss of lock, such as changing PLPRCR[MF] or entering deep-sleep or power-down mode, does not affect SPLSS. 0 SPLL remains locked. 1 SPLL has gone out of lock at least once since the bit was cleared.
17	TEXPS	Timer expired status. Internal status bit 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. This bit is cleared by writing a 1; writing a zero has no effect. When in power-down mode (LPM=11), the TEXPS bit also controls the TEXP external signal as shown below. See Section 14.5.7.1, “Software Initiation of Power-Down Mode, with Automatic Wake-up.” 0 TEXP is negated. 1 TEXP is asserted.
18	—	Reserved, should be cleared.

Table 14-9. PLPRCR Field Descriptions (continued)

Bits	Name	Description
19	TMIST	Timers interrupt status. Cleared at reset. Set when a real-time clock, periodic interrupt timer, timebase, or decrementer interrupt occurs. This bit is cleared by writing a 1; writing a zero has no effect. Entry into low-power mode is disabled when TMIST is set. 0 No timer interrupt was detected. 1 A timer interrupt was detected.
20	—	Reserved, should be cleared.
21	CSRC	Clock source. Specifies whether DFNH or DFNL generates the general system clock. Cleared by hard reset. 0 The general system clock is generated by the DFNH field. 1 The general system clock is generated by the DFNL field.
22–23	LPM	Low-power modes. This bit, in conjunction with TEXPS and CSRC, specifies the operating mode of the core. There are seven possible modes. In the normal modes, the user can write a non-zero value to this field. In the other modes, only a reset or asynchronous interrupt 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.
24	CSR	Checkstop reset enable. 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 Table 14-10. See Section 44.5.2.2, “Debug Enable Register (DER).”
25	LOLRE	Loss-of-lock reset enable. Enables hard reset generation when PLPRCR[SPLSS] is set. 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.
26	FIOPD	Force I/O pull down. 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.
27–31	—	Reserved, should be cleared.

Table 14-10 describes PLPRCR[CSR] and DER[CHSTPE] bit combinations.

Table 14-10. PLPRCR[CSR] and DER[CHSTPE] Bit Combinations

PLPRCR[CSR]	DER[CHSTPE]	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

Chapter 15
Memory Controller

The memory controller is responsible for controlling a maximum of eight memory banks shared between a general-purpose chip-select machine (GPCM) and a pair of sophisticated user-programmable machines (UPMs). 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 the implementation of memory systems with very specific timing requirements.

The GPCM provides interfacing for simpler, lower-performance memory resources and memory-mapped devices. The GPCM has inherently lower performance because it does not support bursting. For this reason, GPCM-controlled banks are used primarily for boot-loading and access to nonburstable memory-mapped peripherals.

The UPM provides both more features and, because it supports bursting, higher performance. Therefore it is typically used to interface with higher-performance run-time memory such as DRAM and bursting SRAM.

The UPM supports address multiplexing of the external bus, periodic timers, and generation of programmable control signals 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.

Different timing patterns can be generated 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. Periodic timers are also available to periodically generate user-defined refresh cycles.

15.1 Features

The following is a list of the memory controller’s main features:

- Eight memory banks
  - 32-bit address decode with mask
  - Variable block sizes (32 Kbytes to 4 Gbytes)
  - Byte parity generation/checking
  - Write-protection capability

  - Address types protection for memory bank accesses by internal masters
  - Control signal generation machine selection on a per-bank basis
  - Support for external master access to memory banks
  - Synchronous and asynchronous external masters support
- General-purpose chip-select machine (GPCM)
  - 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
  - Minimum two clock accesses to external device
  - Four byte write enable signals ($\overline{WE}$)
  - Output enable signal ($\overline{OE}$)
- Two user-programmable machines
  - Programmable-array-based machine controls external signal timing with a granularity of one quarter of an external bus clock period
  - User-specified control-signal patterns run when an internal or external synchronous master requests a single-beat or burst read or write access.
  - User-specified control-signal patterns run when an external asynchronous master requests a single-beat read or write access.
  - UPM periodic timer runs a user-specified control signal pattern to support refresh
  - User-specified control-signal patterns can be initiated by software
  - Each UPM can be defined to support DRAM devices with depths of 64, 128, 256, and 512 Kbytes, and 1, 2, 4, 8, 16, 32, 64, 128, and 256 Mbytes
  - Each UPM provides programmable timing for the following signals:
    - Four byte-select lines
    - Six external general-purpose lines
  - 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 devices
  - Internal address multiplexing for all on-chip bus masters supporting 64-, 128-, 256-, and 512-Kbyte, and 1-, 2-, 4-, 8-, 16-, 32-, 64-, 128-, 256-Mbyte page banks
  - Glueless interface to EDO, self refresh, and synchronous DRAM devices

Figure 15-1 is a block diagram of the memory controller.

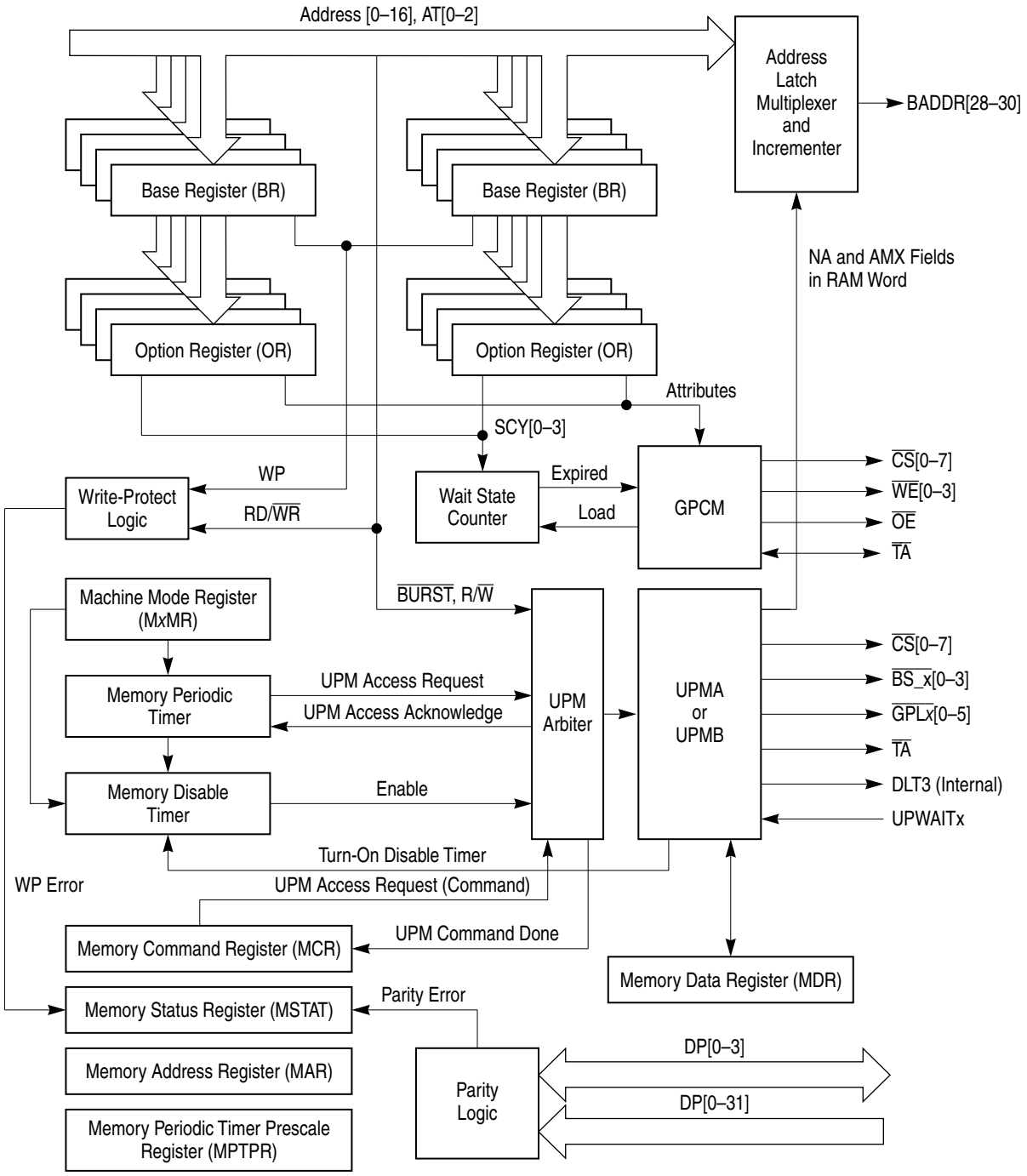


Figure 15-1. Memory Controller Block Diagram

15.2 Basic Architecture

The memory controller consists of three basic machines:

- General-purpose chip-select machine (GPCM)
- User-programmable machine A (UPMA)
- User-programmable machine B (UPMB)

Each bank can be assigned to any one of these machines via the BR*x*[MS] bits as shown in Figure 15-2. Address decode is performed by the comparison of (A[0–16] bit-wise and OR*x*[AM]) with BR*x*[BA]. If an address match occurs in multiple banks, the lowest numbered bank has priority. When a memory address matches BR*x*[BA], the corresponding machine takes ownership of the external signals that control access until the cycle ends.

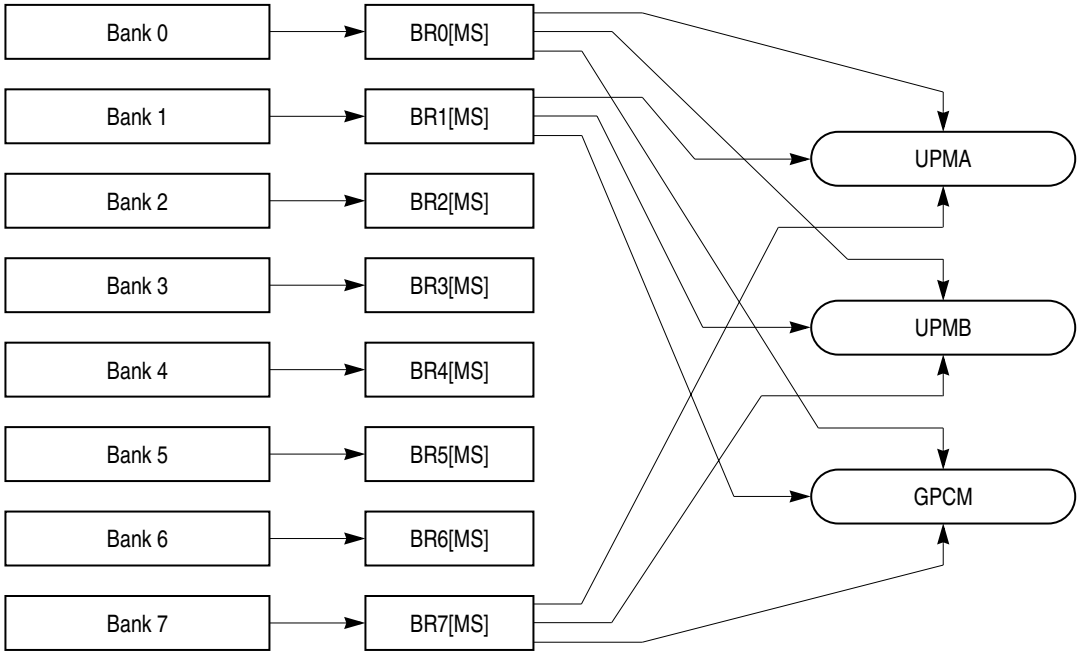


Figure 15-2. Memory Controller Machine Selection

The GPCM provides a glueless interface to EPROM, SRAM, flash EPROM, and other peripherals. GPCM signals are available on $\overline{\text{CS}}$[0–7]. $\overline{\text{CS0}}$ lets the CPU access the boot EPROM from reset. Each chip-select allows up to 30 wait states.

Some features are common to all eight memory banks:

- The block size of each memory bank can vary between 32 Kbytes and 256 Mbytes for a full 4 Gbytes of the address space.
- Parity can be generated and checked for any memory bank. The memory controller has four parity signals (DP[0–3]), one for each data byte lane on the system bus. The parity on the bus is checked only 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 (SIUMCR), which is explained in Section 10.4.2, “SIU Module Configuration Register (SIUMCR).”

- Each memory bank can be selected for read-only or read/write operation.
- For system protection, access to a memory bank can be restricted to accesses with certain address type codes (AT[0–2]). For additional flexibility, address-type comparisons provide a mask option.

The memory controller functionality minimizes the need for glue logic in MPC855T-based systems. In Figure 15-3, $\overline{CS0}$ is used with the 16-bit boot EPROM with BR0[MS] defaulting to select the GPCM. $\overline{CS1}$ is used as the $\overline{RAS}$ signal for 32-bit DRAM with BR1[MS] configured to select UPMA. The $\overline{BS\_A}$ signals are used as $\overline{CAS}$ signals on the DRAM.

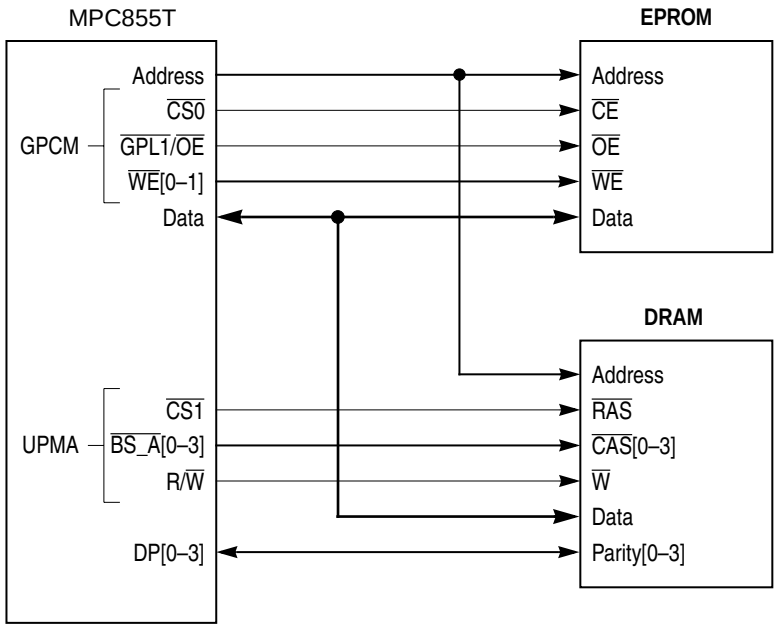


Figure 15-3. Simple System Configuration

The UPMs 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 $\overline{GPL}$ signals. Each memory bank can be assigned to either UPM.

Each UPM is a programmable RAM-based machine. The UPM toggles the memory controller external signals as programmed in RAM when an internal or external master initiates an external single-beat or burst read/write access. The UPM also controls address multiplexing, address increment, and transfer acknowledge assertion for each memory access. The UPM specifies a set of signal patterns for a user-specified number of clock cycles. The UPM RAM pattern run by the memory controller is selected according to the

type of external access transacted. At every clock cycle, the logical value of the external signals specified in the RAM array is output on the corresponding UPM pins. See Figure 15-4.

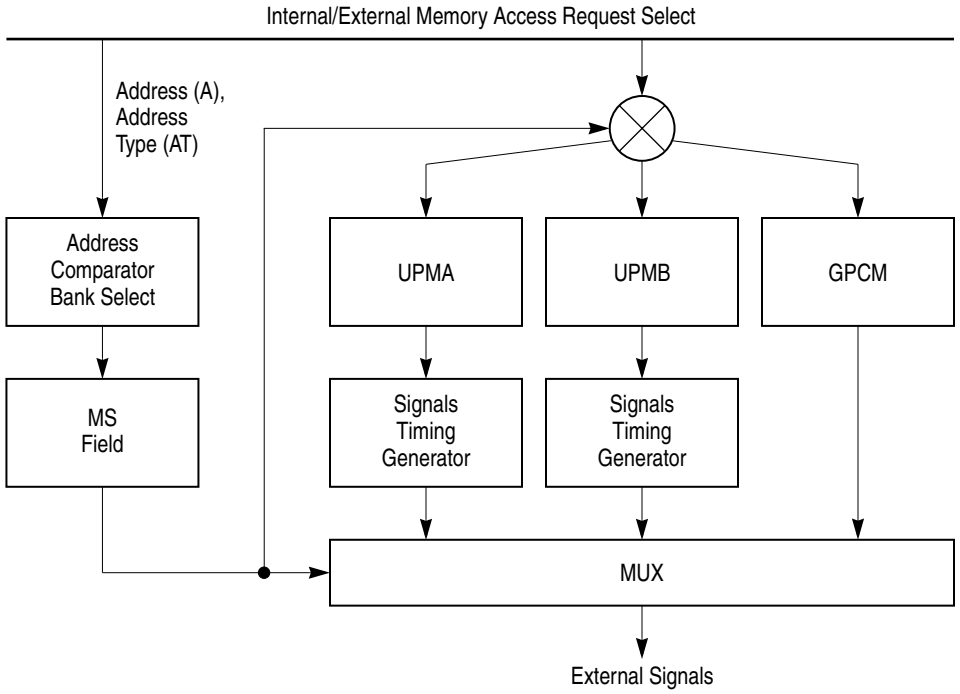


Figure 15-4. Basic Memory Controller Operation

15.3 Chip-Select Programming Common to the GPCM and UPM

The GPCM and the UPMs use the memory controller registers as specified in Table 15-1. See Section 15.4, “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 (BR*x*)	√	√
Option register bank 0–7 register (OR*x*)	√	√
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)		√

15.3.1 Address Space Programming

Each bank has an option register (OR*x*) and a base register (BR*x*), which contains a V bit that indicates that 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. Option registers also define the initial address multiplexing for a memory cycle controlled by a UPM. Each time an internal or external bus cycle access is requested, the address and its corresponding address type are compared to each bank. If one bank matches, its attributes defined in BR*x* and OR*x* are used to control the memory access. If multiple matches occur, the lowest numbered matched bank handles the access.

15.3.2 Register Programming Order

For UPM-controlled chip selects, UPM registers should be programmed before OR*x* and BR*x*. For all chip selects, OR*x* should be programmed before BR*x* except when programming the boot chip select ($\overline{\text{CS0}}$) after hardware reset, in which case, BR0 should be programmed before OR0.

15.3.3 Memory Bank Write Protection

Attempting to write to an address range marked restricted in BR*x*[WP] causes a write-protect violation for which MSTAT[WPER] is set.

15.3.4 Address Type Protection

BR*x*[AT] and OR*x*[ATM] can be used to implement address-type protection in a manner similar to the address space programming. Note that when external masters access memory controller-managed slaves on the bus, the internal AT[0–2] signals to the memory controller are forced to 0b100.

15.3.5 8-, 16-, and 32-Bit Port Size Configuration

The port size is specified by BR*x*[PS]. Eight-bit ports must be connected to D[0–7], 16-bit ports must be connected to D[0–15]. For ports smaller than 32-bits, dynamic bus sizing is performed for all internal masters, such that only external bus accesses result, such as those defined in Table 15-2.

Table 15-2. Access Granularities for Predefined Port Sizes

Predefined Port Size	Bytes		Half Words		Words (on Word Boundaries)
	Odd	Even	Odd	Even	
8-bit	√	√	—	—	—
16-bit	√	√	—	√ (on D[0–15])	—
32-bit	√	√	√	√	√

15.3.6 Parity Configuration

If BR*x*[PARE] is set, parity is generated and checked (for internal masters only) on a per-byte basis using DP[0–3] for the bank. As described in Section 10.4.2, “SIU Module Configuration Register (SIUMCR),” SIUMCR[OPAR] determines the type of parity. Any parity error causes an internal transfer error indication to be asserted and the associated MSTAT[PER] and the corresponding TESR[DPB] or TESR[IPB] to be set, as described in Section 10.4.4, “Transfer Error Status Register (TESR).”

15.3.7 Memory Bank Protection Status

The memory controller status register (MSTAT) reports write-protect violations and parity errors for all eight banks. This protection provided through BR*x*[WP], is intended for detection of erroneous accesses made by DMA from peripherals. More sophisticated protection is provided for accesses from the core by the MMU, as described in Chapter 8, “Memory Management Unit.”

15.3.8 UPM-Specific Registers

The machine *x* mode registers (M*x*MR) define most of the global features for UPMs. The memory command and memory data registers (MCR and MDR) are used to initialize the UPM’s RAM array. MCR[MAD] is the index into the 64-word RAM array for the MDR.

The memory address register (MAR) specifies the address to be driven on the external bus when a UPM pattern is software-initiated by issuing a RUN command in the MCR.

The memory periodic timer prescaler register (MPTPR) defines the divisor of the external bus clock used as the memory periodic timer input.

15.3.9 GPCM-Specific Registers

There are no GPCM-specific registers. All GPCM characteristics are defined in the subfields of individual BR*x* and OR*x* registers.

15.4 Register Descriptions

The following sections describe the registers used by the memory controller.

15.4.1 Base Registers (BR*x*)

The base registers (BR0–BR7) contain the base address and address types that the memory controller uses to compare the value on the address bus with the current address accessed. It also includes a memory attribute and selects the machine for memory operation handling. Figure 15-5 shows the BR*x* register.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	BA															
Reset	xxxx_xxxx_xxxx_xxxx															
R/W	R/W															
Addr	(IMMR & FFFF0000) + 0x100 (BR0), 0x0x108 (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		—					V
Reset	xxxx_xxxx_xx00_0000															
R/W	R/W															
Addr	(IMMR & FFFF0000) + 0x102 (BR0), 0x10A (BR1), 0x112, (BR2), 0x11A (BR3), 0x122 (BR4), 0x12A (BR5), 0x132 (BR6), 0x13A (BR7)															

Figure 15-5. Base Registers (BR*x*)

After reset, BR0 has different default values than other BR*x* registers until the first write to OR0.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	BA															
Reset	xxxx_xxxx_xxxx_xxxx [1]															
R/W	R/W															
Addr	(IMMR & FFFF0000) + 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		—					V
Reset	x	xxx			*		0	0	00		00_000					**
R/W	R/W															
Addr	(IMMR & FFFF0000) + 0x102															

* The reset value of PS depends on the boot port size (BPS) field of the hard reset configuration word.
** The reset value of V depends on the boot disable (BDIS) field of the hard reset configuration word.

Figure 15-6. BR0 Reset Defaults

[1] Becasue at reset the base address value of BR0 is unknown, to ensure proper operation, program BR0 before OR0.

These registers are affected by $\overline{\text{HRESET}}$ but are not affected by $\overline{\text{SRESET}}$. Table 15-3 describes BR*x* fields.

Table 15-3. BR*x* Field Descriptions

Bits	Name	Description
0–16	BA	Base address. Compared to A[0–16]to determine if a memory bank controlled by the memory controller is being accessed by an internal or external bus master. Used in conjunction with OR*x*[AM].
17–19	AT	Address type. Can be used to limit accesses to the memory bank to a certain address space type, AT[0–2]. Note that for internal bus masters, AT[0–2] are sampled from the bus. For external bus masters, AT[0–2] are not sampled on the external bus and instead default to 0b100. Used in conjunction with the OR*x*[ATM].
20–21	PS	Port size. Specifies the port size of the memory region. After system reset, the value of BR0[PS] depends on BPS in the hard reset configuration word, described in Section 11.3.1.1. 00 32-bit port size. 01 8-bit port size. 10 16-bit port size. 11 Reserved.
22	PARE	Parity enable. Used to enable parity checking on this bank. 0 Parity checking is disabled. 1 Parity checking is enabled.
23	WP	Write-protect. Can be used to restrict write accesses within the address range of a BR. 0 Both read and write accesses are allowed. 1 Only read accesses are allowed. The memory controller does not assert $\overline{\text{CS}}$x and $\overline{\text{TA}}$ on write cycles to this memory bank. Attempting to write to the memory bank causes MSTAT[WPER] to be set. The write access is not terminated by the memory controller; however, it is terminated by a $\overline{\text{TEA}}$ assertion from the bus monitor if the bus monitor is enabled.
24–25	MS	Machine select. Selects the machine for handling memory operations. 00 GPCM. 01 Reserved. 10 UPMA. 11 UPMB.
26–30	—	Reserved, should be cleared.
31	V	Valid. Indicates that the contents of the BR*x* and OR*x* are valid. The reset value of BR0[V] depends on BDIS in the hard reset configuration word, described in Section 11.3.1.1. 0 This bank is invalid. An attempt to access this region can cause a bus monitor timeout. 1 This bank is valid. The $\overline{\text{CS}}$ signal does not assert until V is set.

15.4.2 Option Registers (OR*x*)

The option registers (OR0–OR7), shown in Figure 15-7, contain the address and address type mask bit for address bus comparison. It also includes all GPCM parameters.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	AM															
Reset	xxxx_xxxx_xxxx_xxxx															
R/W	R/W															
Addr	(IMMR & FFFF0000) + 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	—
Reset	xxxx_xxxx_xxxx_xxx0															
R/W	R/W															
Addr	(IMMR & FFFF0000) + 0x106															

Figure 15-7. Option Registers (OR*x*)

At reset, OR0 has specific default values and is read-only, as shown in Figure 15-8. After reset, OR0 becomes R/W.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	AM															
Reset	0000_0000_0000_0000															
R/W	R															
Addr	(IMMR & FFFF0000) + 0x104 (OR0)															

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	—
Reset	0	000			1	11		1	1111				0	1	0	0
R/W	R															
Addr	(IMMR & FFFF0000) + 0x106															

Figure 15-8. OR0 Reset Defaults

These registers are affected by HRESET but are not affected by SRESET. Figure 15-4 describes OR*x* fields.

Table 15-4. OR*x* Field Descriptions

Bits	Name	Description
0–16	AM	Address mask. This read/write field independently masks bits A[0–16] on the address bus so external devices of different size address ranges can be used. AM bits can be set or cleared in any order, allowing a resource to reside in more than one area of the address map. 0 The corresponding address bit is masked. 1 The corresponding address bit is used in address pin comparison.
17–19	ATM	Address type mask. Masks certain bits in address type, AT[0–2], 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. If address-type protection is not desired, then ATM should be cleared.
20	CSNT	CSNT (chip-select negation time). Used for the GPCM with ACS and TRLX to control negation of $\overline{\text{CSx}}$ and $\overline{\text{WEx}}$ during an external memory write access. Provides extended address/data hold time for slower memories and peripherals. This will not be applicable when SETA = 1. See Table 15-11.
	SAM	Start address multiplex. Used for a UPM to determine the address output on the first cycle of an external memory access. Should be set only if address multiplexing is to be performed internally. 0 Address pins are not multiplexed internally. 1 Address pins reflect the address requested by the internal master multiplexed according to the setting of MAMR[AMA] (UPMA) or MBMR[AMB] (UPMB).
21–22	ACS	ACS (address to chip-select setup). Lets the GPCM control $\overline{\text{CSx}}$ assertion relative to address lines valid. 00 $\overline{\text{CS}}$ is output at the same time as the address lines. 01 Reserved. 10 $\overline{\text{CS}}$ is output a quarter of a clock after the address lines. 11 $\overline{\text{CS}}$ is output half a clock after the address lines.
	G5LA, G5LS	G5LA and G5LS (general-purpose line 5 A/line 5 start) are used for the UPM to determine how the internal controls and timing generator signal outputs $\overline{\text{GPL5}}$ when a UPM handles a memory access. G5LA (valid only for UPMB): 0 Output the internal $\overline{\text{GPL5}}$ signal on $\overline{\text{GPL\_B5}}$. 1 Output the internal $\overline{\text{GPL5}}$ signal on $\overline{\text{GPL\_A5}}$.
		G5LS (valid for UPMA or UPMB) 0 $\overline{\text{GPL5}}$ is driven low on the falling edge of GCLK1_50 in the first clock cycle of a memory access. 1 $\overline{\text{GPL5}}$ is driven high on the falling edge of GCLK1_50 in the first clock cycle of a memory access.
23	BIH	Burst inhibit. Determines whether this memory bank supports burst accesses. If the machine selected to handle this access is the GPCM, BIH must be set. 0 $\overline{\text{BI}}$ is negated. The bank supports burst accesses. 1 $\overline{\text{BI}}$ is asserted. The bank does not support burst accesses.
24–27	SCY	Select cycle length (GPCM only). Binary representation of the number of wait states inserted in the cycle when the GPCM handles an external memory access (0000 = 0 clock cycle, 0001 = 1 clock cycle, …, 1111 = 15 clock cycle). Total cycle length is also affected by TRLX. See Table 15-11 for the total number of cycles. If external $\overline{\text{TA}}$ response is selected (SETA = 1), SCY is not used.
28	SETA	Select external transfer acknowledge (GPCM only). 0 Internal or external transfer acknowledge can acknowledge this access, whichever comes first. 1 The memory controller does not generate $\overline{\text{TA}}$ for this bank; instead the peripheral must generate it on the external $\overline{\text{TA}}$ signal.

Table 15-4. ORx Field Descriptions (continued)

Bits	Name	Description
29	TRLX	Timing relaxed (GPCM only) 0 Timing is not relaxed. 1 In addition to the timing parameters programmed in other ORx fields, timing is further relaxed. See the effect of TRLX in Table 15-11. TRLX also doubles the wait-states programmed in SCY.
30	EHTR	Extended hold time on read. (GPCM only) 0 Timing is defined by the memory controller. 1 After a read from the current bank, an additional clock cycle is inserted before the memory controller responds to a write or read to another bank.
31	—	Reserved, should be cleared.

15.4.3 Memory Status Register (MSTAT)

The memory status register (MSTAT) reports parity and write-protect errors encountered during an external bus access initiated by the memory controller. Writing ones to specific bits clears them; writing zeros has no effect.

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	—						
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	(IMMR & FFFF0000) + 0x178															

Figure 15-9. Memory Status Register (MSTAT)

This register is affected by $\overline{\text{HRESET}}$ but is not affected by $\overline{\text{SRESET}}$. Table 15-5 describes MSTAT fields.

Table 15-5. MSTAT Field Descriptions

Bits	Name	Description
0–7	PERx	Parity error bank 0–7. Set when a parity error is detected during a read cycle to this bank initiated by the memory controller.
8	WPER	Write-protection error. Set when a write-protect error occurs on a write cycle to a write-protected bank defined by BRx[WP].
9–15	—	Reserved, should be cleared.

15.4.4 Machine A Mode Register/Machine B Mode Registers (MxMR)

The machine *x* mode register (MAMR and MBMR) contain the configuration for UPMA and UPMB, respectively. See Figure 15-1.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	PT*x*								PT*x*E	AM*x*			—	DS*x*		—
Reset	xxxx_xxxx_0000_0000															
R/W	R/W															
Addr	(IMMR & FFFF0000) + 0x170															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	G0CL*x*			GPL*x*4DIS	RLF*x*				WLF*x*				TLF*x*			
Reset	000			1	0000				0000				0000			
R/W	R/W															
Addr	(IMMR & FFFF0000) + 0x172															

Figure 15-10. Machine A Mode Register/Machine B Mode Register (M*x*MR)

This register is affected by $\overline{\text{HRESET}}$ but is not affected by $\overline{\text{SRESET}}$. Table 15-6 describes bits for MAMR/MBMR.

Table 15-6. M*x*MR Field Descriptions

Bits	Name	Description
0–7	PT*x*	Periodic timer *x* period. Affects periodic timer *x* and determines the timer period service rate according to the following equation, which determines value for UPM*x* to refresh memory: $\text{PTx} = \frac{\text{System Clock (MHz)} \times \text{Service Duration (µs)}}{2^{2 \times \text{SCCR[DFBRG]}} \times \text{Prescaler (PTP)} \times \text{NCS}}$ NCS is an integer between 1 and 8 that represents the number of enabled chip selects that are serviced by this UPM. SCCR[DFBRG] is defined in Section 14.6.1, “System Clock and Reset Control Register (SCCR).” For example, for DRAM to maintain data integrity, an access or refresh must occur every 15.6 µs. Given a 25-MHz system clock with the required service rate of 15.6µs, a periodic timer prescaler = 32, and DFBRG = 0, PT*x* = (25 x 15.6) / (2^{2x0} x 32x 1) = 12.
8	PT*x*E	Periodic timer *x* enable. Allows the periodic timer *x* to request service. 0 Periodic timer *x* is disabled. 1 Periodic timer *x* is enabled.
9–11	AM*x*	Address multiplex size *x*. When internal address multiplexing is used, this field specifies how the address on the external bus is multiplexed, when enabled (see Table 15-18). The SAM bit enables address multiplexing in the first clock cycle. The AM*x* field of the RAM array entry enables address multiplexing in subsequent clock cycles. (see Table 15-19).
12	—	Reserved, should be cleared.

Table 15-6. M*x*MR Field Descriptions (continued)

Bits	Name	Description
13–14	DS*x*	Disable timer period. Guarantees a minimum time between accesses to the same memory bank if it is controlled by the UPM*x*. This function can be used to guarantee a minimum $\overline{\text{RAS}}$ precharge time. The TODT bit in the RAM array turns on the disable timer and, when expired, the UPM*x* allows the machine access to issue a memory pattern to the same region. An access attempted before the timer expires (as signalled by $\overline{\text{TS}}$ assertion) has wait states inserted before the UPM pattern runs. Accesses to other chip-selects serviced by this UPM are unaffected by this timer. The maximum disable period is four clock cycles. If more than 4 cycles are required, they must be added explicitly in the UPM RAM words. 00 1-cycle disable period 01 2-cycle disable period 10 3-cycle disable period 11 4-cycle disable period
15	—	Reserved, should be cleared.
16–18	G0CL*x*[0–2]	General line 0 control *x*. Selects the address line output to the internal $\overline{\text{GPL0}}$ signal in the special case where the functionality is enabled in the G0L and G0H bits of the UPM RAM word. 000 = A12 100 = A8 001 = A11 101 = A7 010 = A10 110 = A6 011 = A9 111 = A5
19	GPL*x*4DIS	GPL*x*4 output line disable. Determines whether UPWAIT*x*/$\overline{\text{GPL}\_x4}$ behaves as the $\overline{\text{GPL4}}$ output controlled the UPM RAM word, or an input signal used to dynamically insert wait states into UPM patterns. 0 = UPWAIT*x*/$\overline{\text{GPL}\_x4}$ is defined as $\overline{\text{GPL}\_x4}$. 1 = UPWAIT*x*/$\overline{\text{GPL}\_x4}$ is defined as UPWAIT*x*.
20–23	RLF*x*	Read loop field *x*. Specifies (in binary) the number of times a loop defined in the UPM*x* RAM word is executed for a burst read or single-beat read cycle. (0001 = 1 time, 0010 = 2 times, …, 1111 = 15 times; note that 0000 = 16 times.)
24–27	WLF*x*	Write loop field *x*. Specifies the number of times a loop defined in the UPM*x* RAM word is executed for a burst- or single-beat write cycle. (0001 = 1 time, 0010 = 2 times, …, 1111 = 15 times; note that 0000 = 16 times.)
28–31	TLF*x*	Timer loop field *x*. Specifies the number of times a loop defined in the UPM*x* RAM word is executed for a periodic timer service. (0001 = 1 time, 0010 = 2 times, …, 1111 = 15 times; note that 0000 = 16 times.)

15.4.5 Memory Command Register (MCR)

The memory command register (MCR) is used during UPM initialization to read and write the contents of the UPM RAM. It also allows commands to be issued that stimulate UPM routine execution. This capability lets the CPU perform special memory operations in addition to standard read/write and periodic timer service operations. An example of this is software execution of a special UPM pattern to initialize SDRAM.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	OP		—						UM	—						
Reset	xx00_0000_x000_0000															
R/W	R/W															
Addr	(IMMR & FFFF0000) + 0x168															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	MB			—	MCLF				—		MAD					
Reset	xxx0_xxxx_00xx_xxxx															
R/W	R/W															
Addr	(IMMR & FFFF0000) + 0x16A															

Figure 15-11. Memory Command Register (MCR)

This register is affected by $\overline{\text{HRESET}}$ but is not affected by $\overline{\text{SRESET}}$. Table 15-7 describes MCR fields.

Table 15-7. MCR Field Descriptions

Bits	Name	Description
0–1	OP	Command opcode. Defines the operation to be executed by the UPM specified in the UM field. 00 WRITE writes the contents of the MDR into the RAM location indexed by MCR[MAD]. 01 READ reads the contents of the RAM location indexed by MCR[MAD] and stores it in the MDR. 10 RUN executes the pattern in the RAM array beginning with the RAM word indexed by MCR[MAD] on the memory bank specified in MCR[MB]. The AMX bits of the UPM RAM word in this software-initiated pattern must all be set to 0b11. Thus, the address for this pattern is the value written to MAR. The data bus is not driven. 11 Reserved.
2–7	—	Reserved, should be cleared.
8	UM	User machine. Selects the UPM for this command. 0 UPMA 1 UPMB
9–15	—	Reserved, should be cleared.
16–18	MB	Memory bank. Indicates the appropriate $\overline{\text{CSx}}$ pin when a run command is executed (000 corresponds to $\overline{\text{CS0}}$, 001 corresponds to $\overline{\text{CS1}}$, …, 111 corresponds to $\overline{\text{CS7}}$)
19	—	Reserved, should be cleared.
20–23	MCLF	Memory command loop field. Specifies how many times a loop is executed for a RUN command. (0001 = the loop executes once, 0010 = the loop executes twice, …, 1111 = the loop executes 15 times. Note that 0000 = the loop executes 16 times.)
24–25	—	Reserved, should be cleared.
26–31	MAD	Memory array index. Specifies an index to one of 64 RAM words in the RAM array.

15.4.6 Memory Data Register (MDR)

The memory data register (MDR) contains data written to or read from the RAM array for UPM READ or WRITE commands. MDR must be set up before issuing a WRITE command to the MCR.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	MD															
Reset	xxxx_xxxx_xxxx_xxxx															
R/W	R/W															
Address	(IMMR & FFFF0000) + 0x17C															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	MD															
Reset	xxxx_xxxx_xxxx_xxxx															
R/W	R/W															
Address	(IMMR & FFFF0000) + 0x17E															

Figure 15-12. Memory Data Register (MDR)

This register is not affected by $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$. Table 15-8 describes MDR.

Table 15-8. MDR Field Descriptions

Bits	Name	Description
0–31	MD	Memory data. Contains the RAM array word.

15.4.7 Memory Address Register (MAR)

The memory address register contains an address to be driven on the external bus in the case of a RUN command issued to the MCR.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	MA															
Reset	xxxx_xxxx_xxxx_xxxx															
R/w	R/W															
Address	(IMMR & FFFF0000) + 0x164															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	MA															
Reset	xxxx_xxxx_xxxx_xxxx															
R/W	R/W															
Address	(IMMR & FFFF0000) + 0x166															

Figure 15-13. Memory Address Register (MAR)

This register is not affected by $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$. Table 15-9 describes MAR fields.

Table 15-9. MAR Field Description

Bits	Name	Description
0–31	MA	Contains a 32-bit address to be output on the address bus if AMX = 0b11. See Section 15.6.4.1, “RAM Words.”

15.4.8 Memory Periodic Timer Prescaler Register (MPTPR)

The memory periodic timer prescaler register (MPTPR) defines the divisor of the external bus clock used as the memory periodic timer input clock. See Section 14.3, “Clock Signals.”

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	PTP								—							
Reset	0000_001x								0000_0000							
R/W	R/W															
Addr	(IMMR & FFFF0000) + 0x17A															

Figure 15-14. Memory Periodic Timer Prescaler Register (MPTPR)

This register is affected by $\overline{\text{HRESET}}$ but is not affected by $\overline{\text{SRESET}}$. Table 15-10 describes MPTPR fields.

Table 15-10. MPTPR Field Descriptions

Bits	Name	Description
0–7	PTP	Periodic timers prescaler. Contains the division factor defined below. 001x xxxxDivide by 2. 0001 xxxxDivide by 4. 0000 1xxxDivide by 8. 0000 01xxDivide by 16. 0000 001xDivide by 32. 0000 0001Divide by 64. All other values are reserved.
8–15	—	Reserved, should be cleared.

15.5 General-Purpose Chip-Select Machine (GPCM)

The GPCM allows a glueless and flexible interface between the MPC855T, SRAM, EPROM, FEPROM, ROM devices, and external peripherals. The GPCM contains three basic configuration register groups—BR*x*, OR*x*, and MSTAT.

The GPCM provides a $\overline{\text{CS}}$ signal for memory bank activation, $\overline{\text{WE}}$ signals for write cycles for each byte written to memory, and $\overline{\text{OE}}$ signals for read cycles. Figure 15-15 shows a simple connection between an SRAM device and the MPC855T.

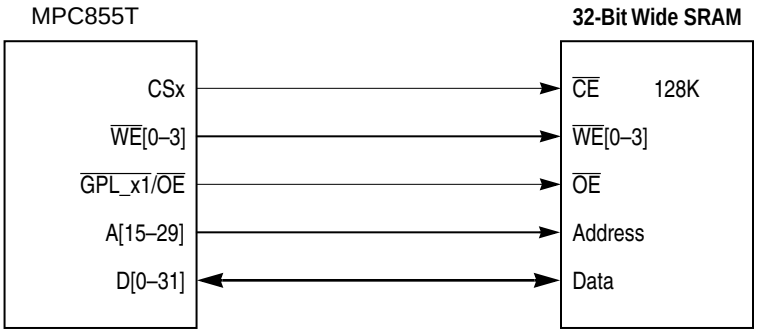


Figure 15-15. GPCM-to-SRAM Configuration

15.5.1 Timing Configuration

If BRx[MS] selects the GPCM, the attributes for the memory cycle are taken from OR*x*. These attributes include the CSNT, ACS[0–1], SCY[0–3], TRLX, EHTR, and SETA fields. See Table 15-11 for signal behavior and system response.

Table 15-11. GPCM Strobe Signal Behavior

Configuration					Signal Behavior						
OR*x* [TRLX]	Access	SCCR [EBDF]	OR*x* [CSNT]	OR*x* [ACS]	Address to $\overline{CS}$ Asserted	Address to $\overline{OE}$ Asserted	Address to $\overline{WE}$ Asserted	Data to $\overline{WE}$ Asserted	$\overline{CS}$ Negated to Address/ Data Invalid	$\overline{WE}$ Negated to Address/ Data Invalid	Total Cycles
0	Read	x	x	00	0	3/4*Clk	x	x	1/4* Clk	x	2+SCY [1]
				10	1/4*Clk						
				11	1/2*Clk						
	Write		0	00	0	x				1/4*Clk	
				10	1/4*Clk		3/4*Clk	-1/4*Clk			
				11	1/2*Clk						
		00	1	00	0					1/2*Clk	
				10	1/4*Clk				1/2*Clk		
				11	1/2*Clk						
		01		00	0				1/4*Clk	3/8*Clk	
				10	1/4*Clk				3/8*Clk		
				11	1/2*Clk						

Table 15-11. GPCM Strobe Signal Behavior (continued)

Configuration					Signal Behavior						
ORx [TRLX]	Access	SCCR [EBDF]	ORx [CSNT]	ORx [ACS]	Address to $\overline{CS}$ Asserted	Address to $\overline{OE}$ Asserted	Address to $\overline{WE}$ Asserted	Data to $\overline{WE}$ Asserted	$\overline{CS}$ Negated to Address/ Data Invalid	$\overline{WE}$ Negated to Address/ Data Invalid	Total Cycles
1	Read	x	x	00	0	3/4*Clk	x	x	1/4*Clk	x	2+2*SCY
				10	1+1/4*Clk	1+3/4*Clk					3+2*SCY
				11	1+1/2*Clk						
	Write		0	00	0	x	3/4*Clk	-1/4*Clk		1/4*Clk	2+2*SCY
				10	1+1/4*Clk		1+3/4*Clk	3/4*Clk			3+2*SCY
				11	1+1/2*Clk						
		00	1	00	0		3/4*Clk	-1/4*Clk		1+1/2*Clk	
				10	1+1/4*Clk		1+3/4*Clk	3/4*Clk	1+1/2*Clk		4+2*SCY
				11	1+1/2*Clk						
		01		00	0		3/4*Clk	-1/4*Clk	1/4*Clk	1+3/8*Clk	3+2*SCY
				10	1+1/4*Clk		1+3/4*Clk	3/4*Clk	1+3/8*Clk		4+2*SCY
				11	1+1/2*Clk						

1 SCY is the number of wait cycles from the option register.

15.5.1.1 Chip-Select Assertion Timing

The banks selected by the GPCM support an option to output $\overline{CS}$ at different timings with respect to the external address bus. Depending on the value of the ACS field (plus an additional cycle if TRLX = 1), $\overline{CS}$ can be output as follows

- Simultaneous with the external address
- One quarter of a clock cycle later
- One half of a clock cycle later

Figure 15-16 shows a basic connection between the MPC855T and an external peripheral device. Here, $\overline{CS}$ (the strobe output for the memory access) is connected directly to $\overline{CE}$ of the memory device and R/$\overline{W}$ is connected to the respective R/$\overline{W}$ in the peripheral device.

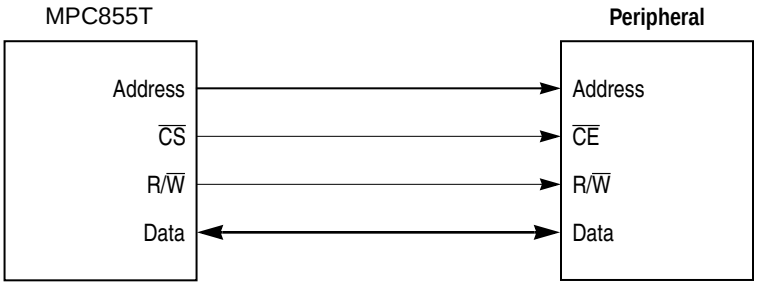


Figure 15-16. GPCM Peripheral Device Interface

Figure 15-17 shows $\overline{CS}$ as defined by the setup time required between the address lines and $\overline{CE}$. The user can configure OR*x*[ACS**]** to specify $\overline{CS}$ to meet this requirement.

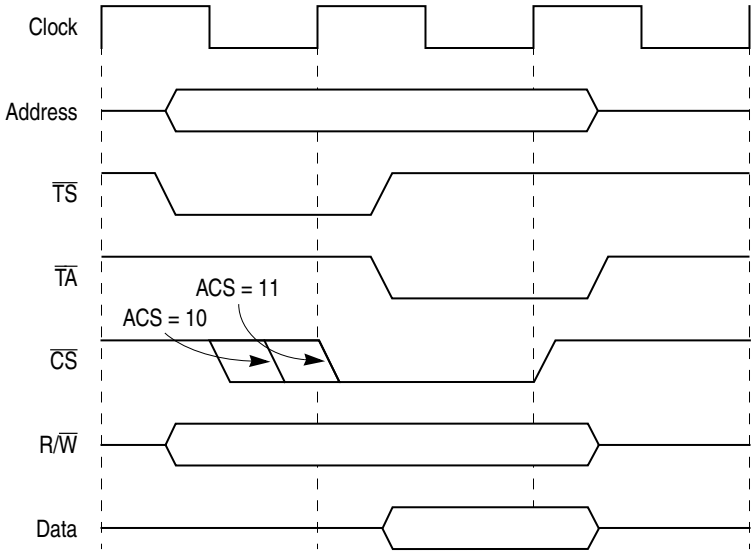


Figure 15-17. GPCM Peripheral Device Basic Timing (ACS = 1x and TRLX = 0)

15.5.1.2 Chip-Select and Write Enable Deassertion Timing

Figure 15-18 shows a basic connection between the MPC855T and a static memory device. Here, $\overline{CS}$ is connected directly to $\overline{CE}$ of the memory device. The $\overline{WE}$ signals are connected to the respective $\overline{W}$ signal in the memory device where each $\overline{WE}$ corresponds to a different data byte.

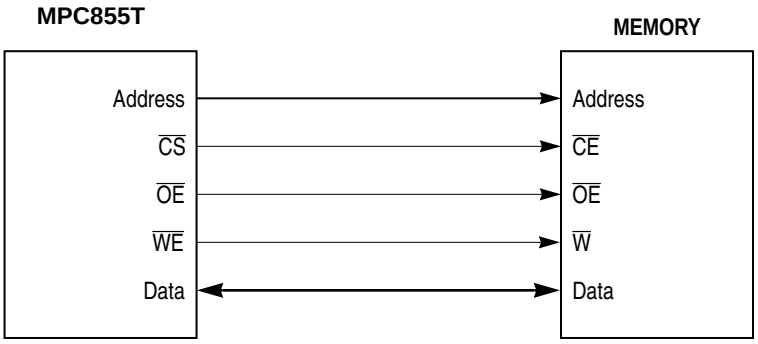


Figure 15-18. GPCM Memory Device Interface

As Figure 15-20 shows, the timing for $\overline{CS}$ is the same as for the address lines. The strobes for the transaction are supplied by $\overline{OE}$ or $\overline{WE}$, depending on the transaction direction (read or write). OR*x*[CSNT] 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 ACS = 00 and CSNT = 1, $\overline{WE}$ is negated one quarter of a clock earlier, as shown in Figure 15-19. When ACS ≠ 00 and CSNT = 1, $\overline{WE}$ and $\overline{CS}$ are negated one quarter of a clock earlier, as shown in Figure 15-20.

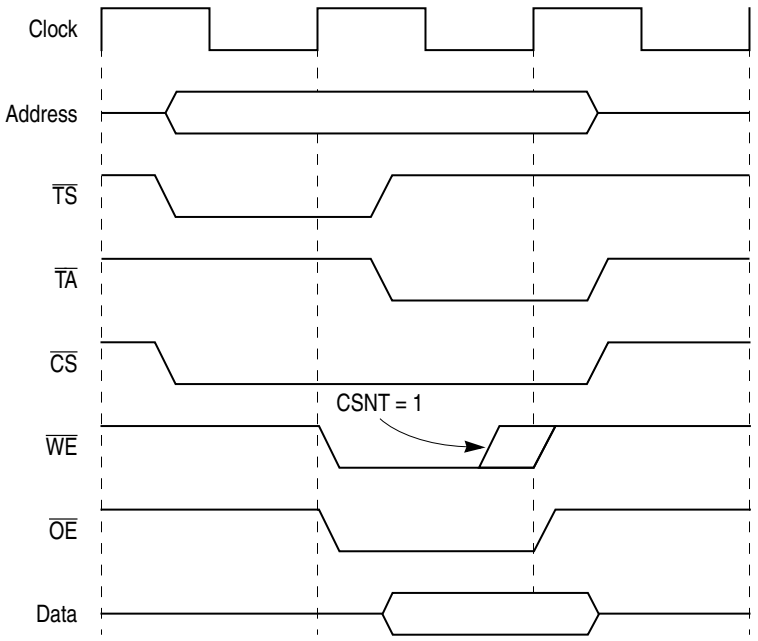


Figure 15-19. GPCM Memory Device Basic Timing (ACS = 00, CSNT = 1, TRLX = 0)

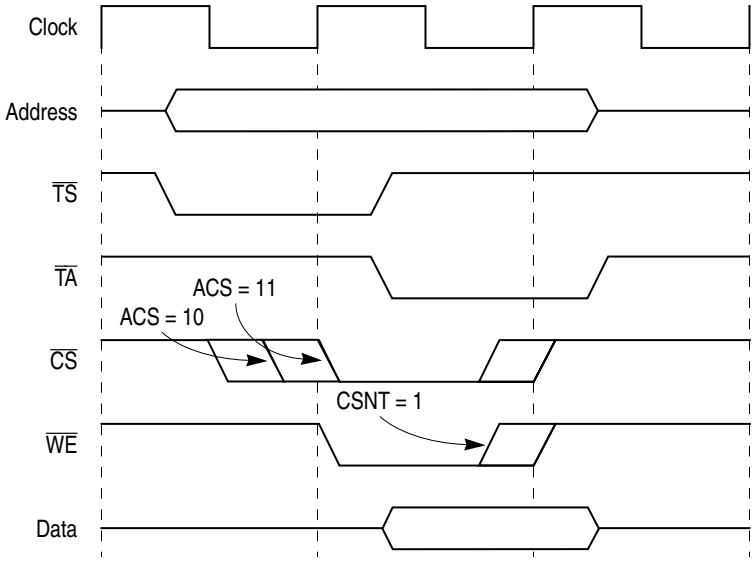


Figure 15-20. GPCM Memory Device Basic Timing (ACS ≠ 00, CSNT = 1, TRLX = 0)

15.5.1.3 Relaxed Timing

OR*x*[TRLX] is provided for memory systems that require more relaxed timing between signals. When TRLX = 1 and ACS ≠ 00, an additional cycle between the address and strobes is inserted by the MPC855T memory controller. See Figure 15-21 and Figure 15-22.

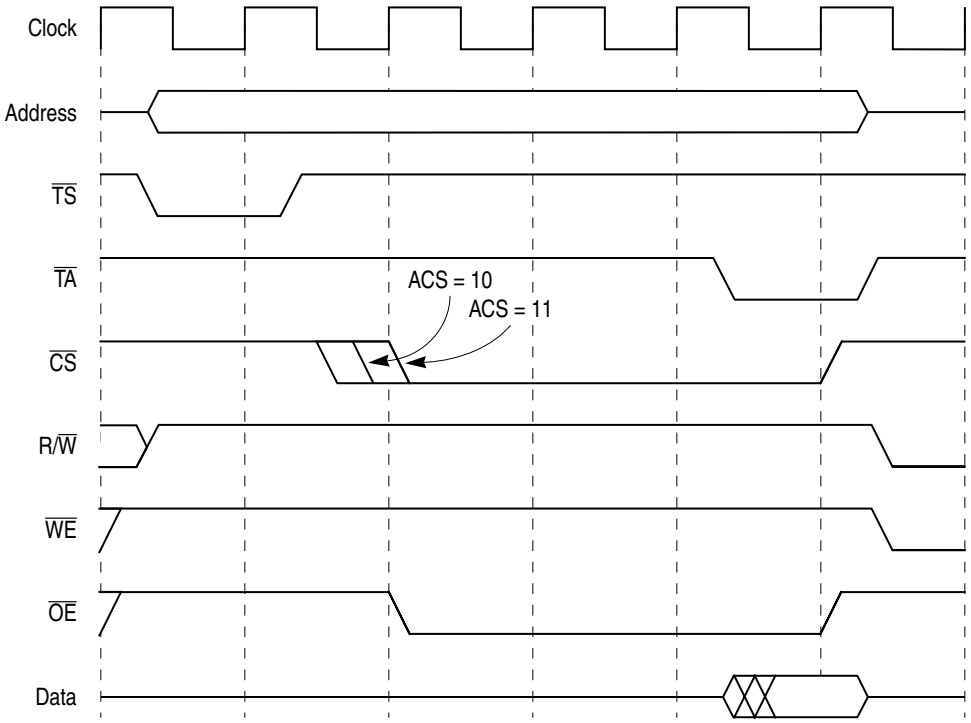


Figure 15-21. GPCM Relaxed Timing Read (ACS = 1x, SCY = 1, CSNT = 0, and TRLX = 1)

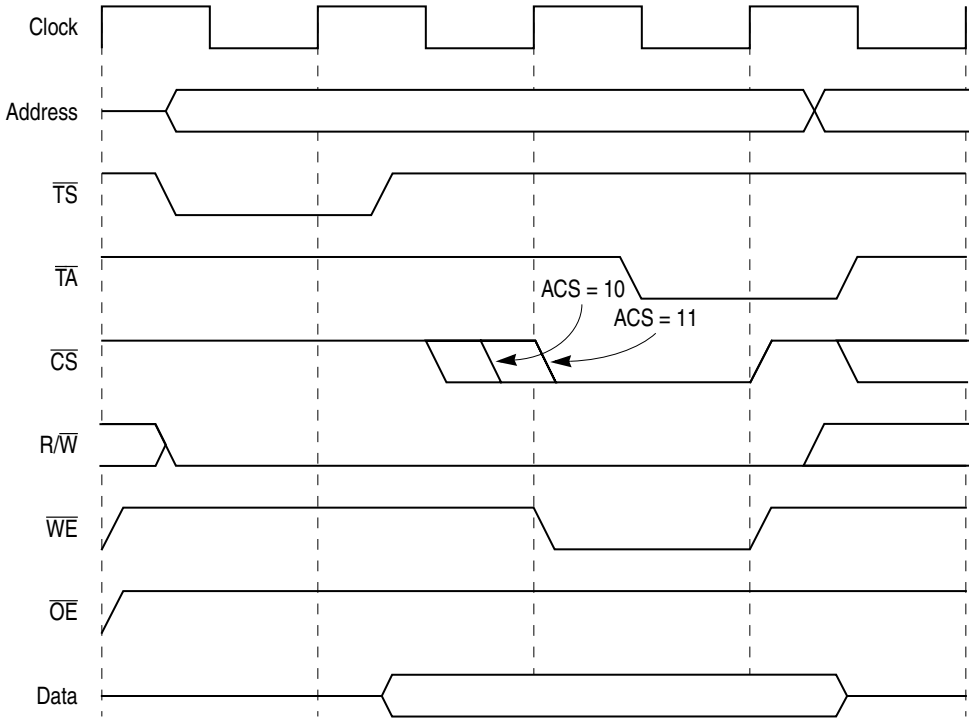


Figure 15-22. GPCM Relaxed-Timing Write (ACS = 1x, SCY = 0, CSNT = 0, TRLX = 1)

When TRLX and CSNT are set in a write-memory access, the strobe line, $\overline{WE}$ is negated one clock earlier than in the normal case. If ACS ≠ 0, $\overline{CS}$ is also negated one clock earlier, as shown in Figure 15-23 and Figure 15-24. When a bank is selected to operate with external transfer acknowledge (SETA and TRLX = 1), the memory controller does not support external devices that provide $\overline{TA}$ to complete the transfer with zero wait states. The minimum access duration in this case is 3 clock cycles.

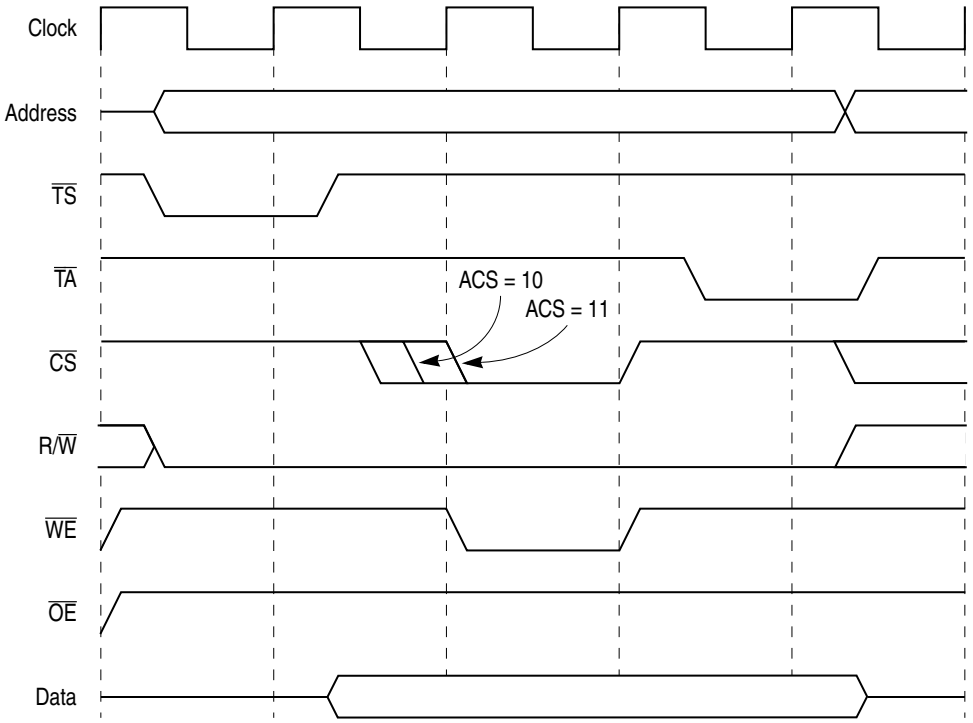


Figure 15-23. GPCM Relaxed-Timing Write (ACS = 1x, SCY = 0, CSNT = 1, TRLX =1)

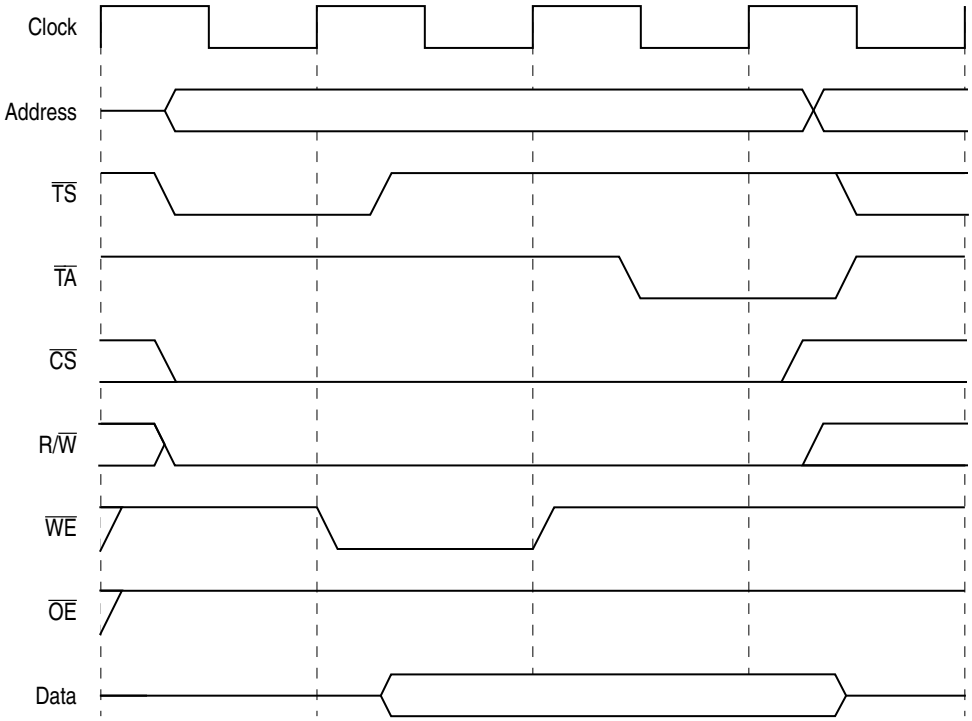


Figure 15-24. GPCM Relaxed-Timing Write (ACS = 00, SCY = 0, CSNT = 1, TRLX =1)

15.5.1.4 Output Enable ($\overline{OE}$) Timing

The timing of the $\overline{OE}$ is affected only by TRLX. It always asserts and negates on the rising edge of the external bus clock. $\overline{OE}$ always asserts on the rising clock edge after $\overline{CS}$ is asserted, and therefore its assertion can be delayed (along with the assertion of $\overline{CS}$) by programming TRLX = 1. $\overline{OE}$ deasserts on the rising clock edge coinciding with or immediately following $\overline{CS}$ deassertion.

15.5.1.5 Programmable Wait State Configuration

The GPCM supports internal $\overline{TA}$ generation. It allows fast accesses to external memory through an internal bus master or a maximum 17-clock access by programming OR*x*[SCY]. The internal $\overline{TA}$ generation mode is enabled if OR*x*[SETA] is cleared. If $\overline{TA}$ is asserted externally at least two clock cycles before the wait state counter has expired, the current memory cycle is terminated. When TRLX 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.

15.5.1.6 Extended Hold Time on Read Accesses

Slow memory devices that take a long time to turn off their data bus drivers on read accesses should set OR*x*[EHTR]. Any MPC855T 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. See Figure 15-25 through Figure 15-28 for details.

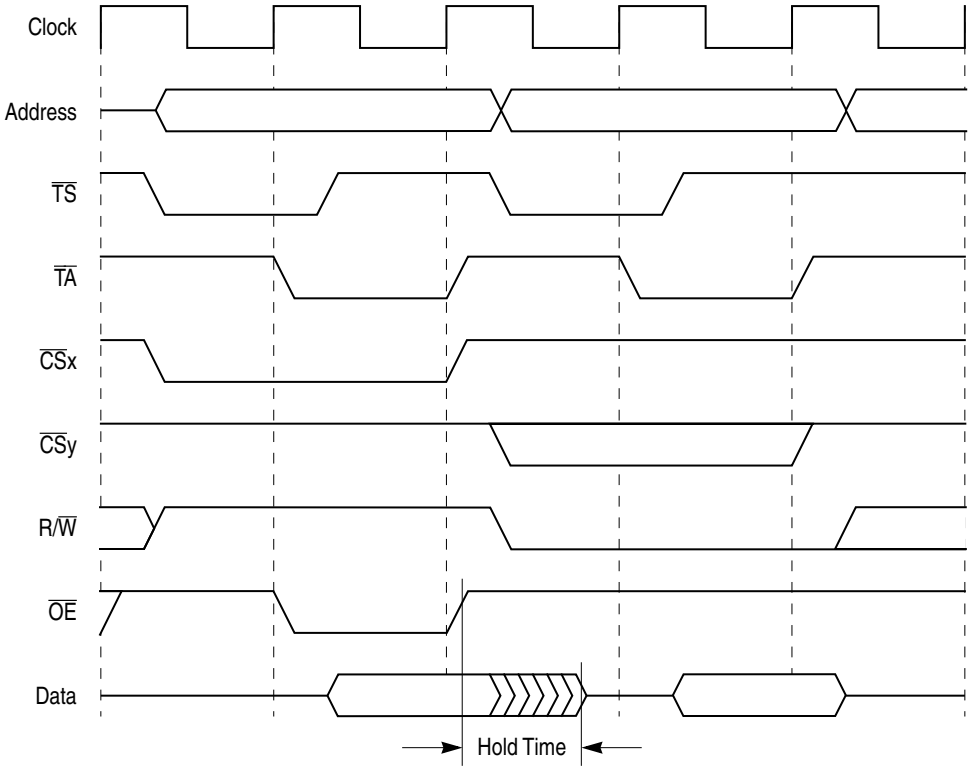


Figure 15-25. GPCM Read Followed by Write (EHTR = 0)

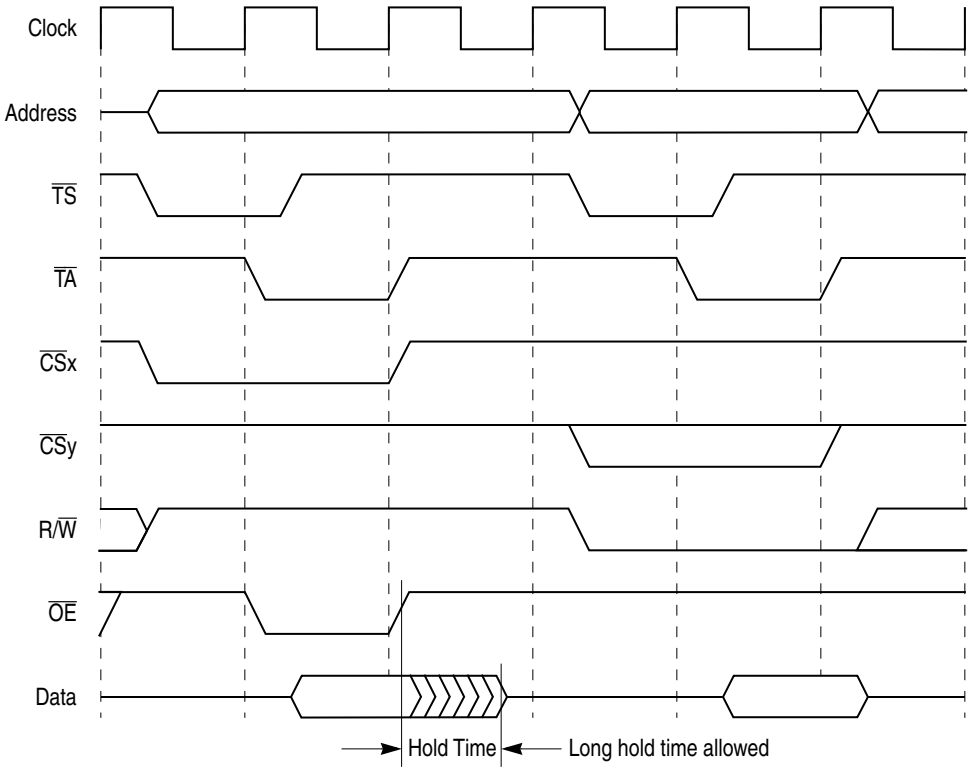


Figure 15-26. GPCM Read Followed by Write (EHTR = 1)

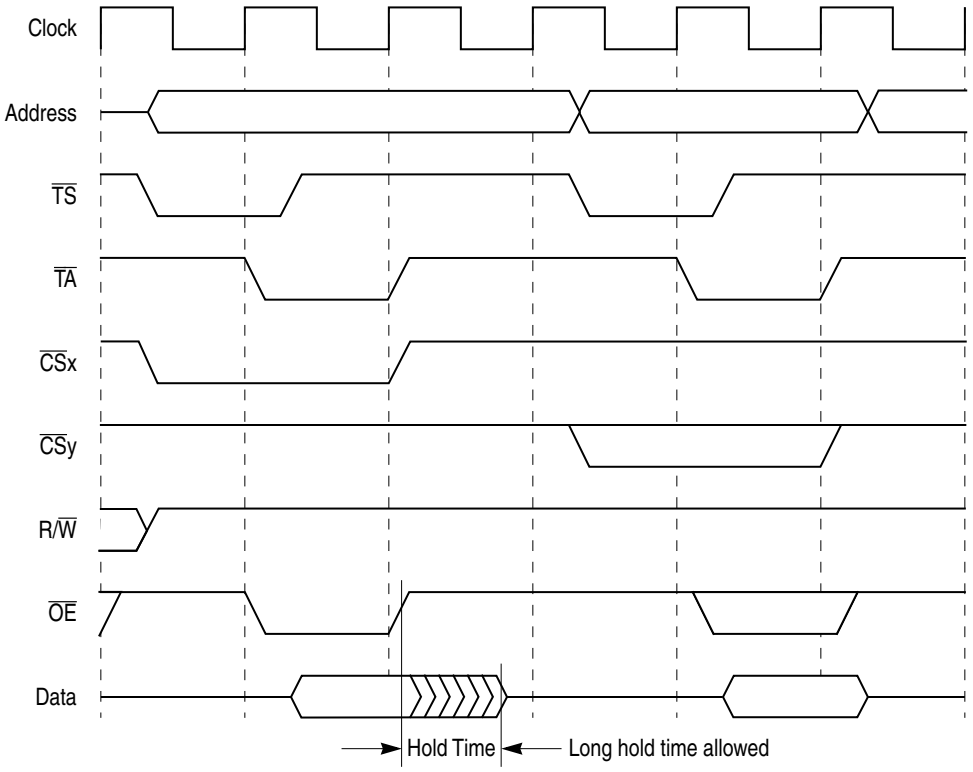


Figure 15-27. GPCM Read Followed by Read from Different Banks (EHTR = 1)

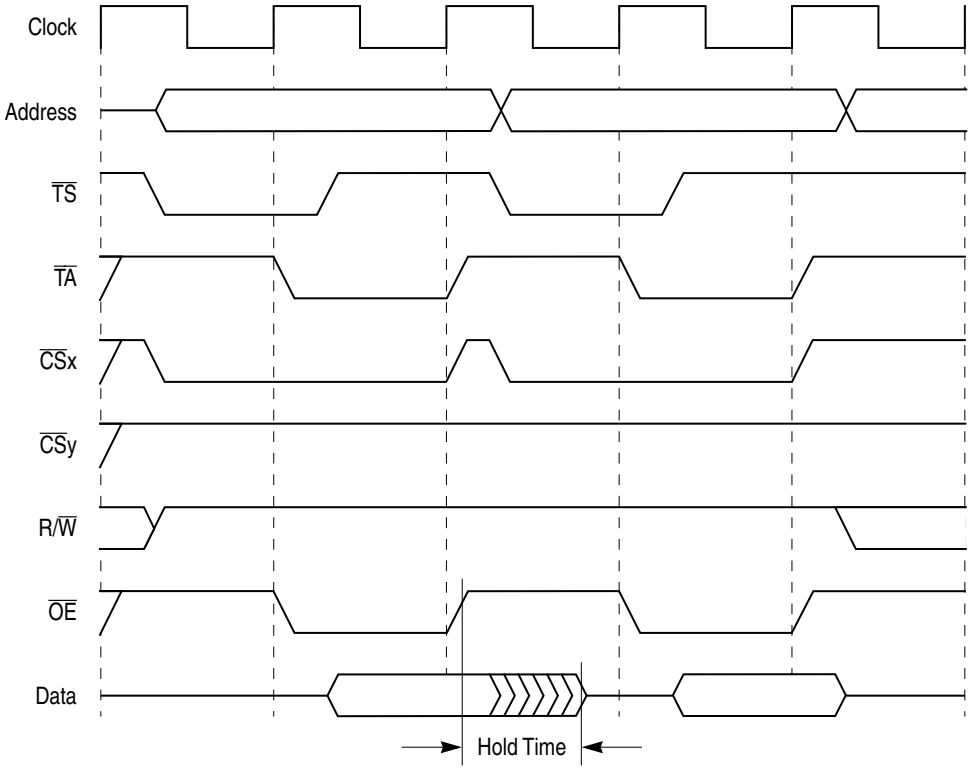


Figure 15-28. GPCM Read Followed by Read from Same Bank (EHTR = 1)

15.5.2 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 MPC855T 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 described in Section 11.3.1.1. 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 OR0 and it can be used as any other chip-select register once the preferred address range is loaded into BR0. After the first write to OR0, the boot chip-select can only be restarted on hardware reset. The initial values of the boot bank in the memory controller are described in Table 15-12.

Table 15-12. Boot Bank Field Values after Reset

Register	Field Name	Value
BR0	PS	From hard reset configuration word
	PARE	0
	WP	0
	MS	00
	V	From hard reset configuration word
OR0	AM	All zeros
	ATM	000
	CSNT	1
	ACS	11
	SCY	1111
	SETA	0
	TRLX	1
	EHTR	0

15.5.3 External Asynchronous Master Support

Figure 15-29 shows the basic interface between an asynchronous external master and the GPCM to allow connection to static RAM.

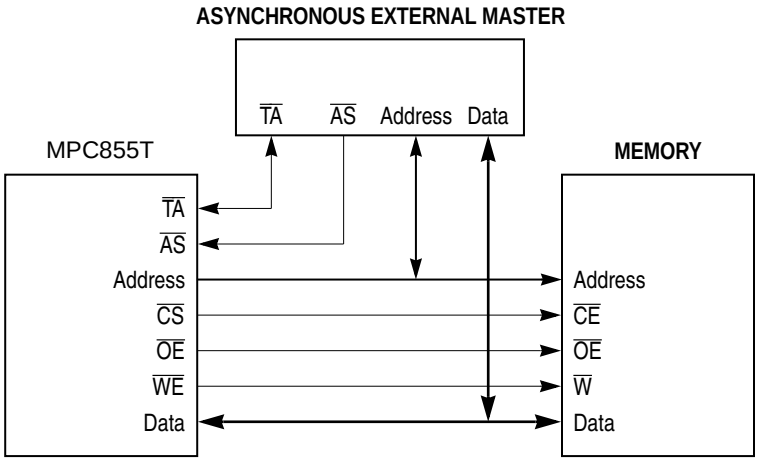


Figure 15-29. Asynchronous External Master Configuration for GPCM-Handled Memory Devices

Figure 15-30 shows the timing for TRLX = 0 when an external asynchronous master accesses SRAM. $\overline{TA}$, $\overline{WE}$, and $\overline{OE}$ remain asserted until the external master negates $\overline{AS}$, at which point they deassert asynchronously.

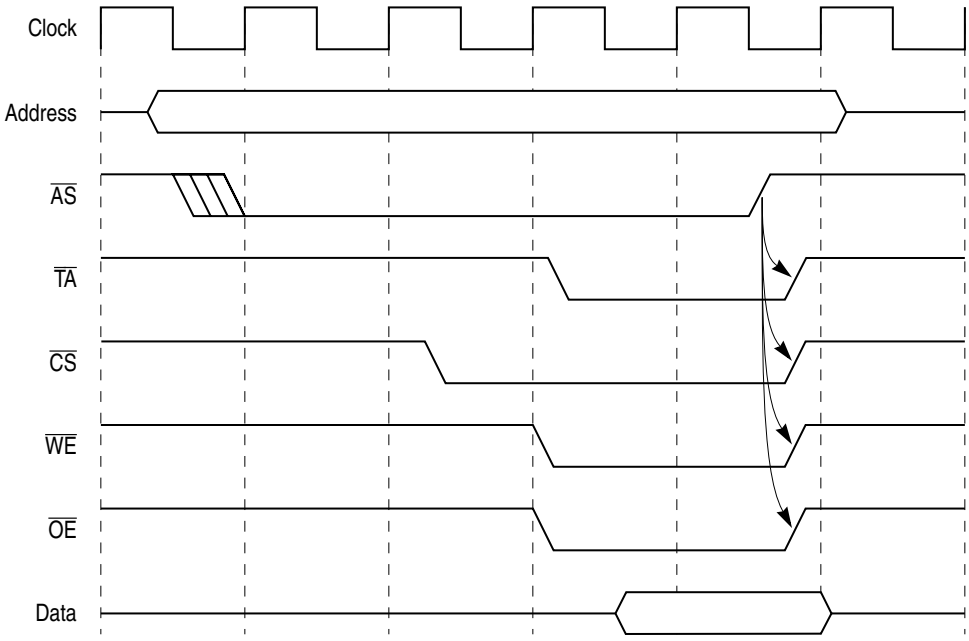


Figure 15-30. Asynchronous External Master, GPCM-Handled Memory Access Timing (TRLX = 0)

When an external asynchronous master performs accesses a memory device via the GPCM in the memory controller, OR*x*[CSNT] has no effect.

For a comprehensive discussion of external master interfacing, see Section 15.8, “External Master Support.”

15.5.4 Special Case: Bursting with External Transfer Acknowledge:

The memory controller supports bursting to and from an external slave that supplies its own $\overline{TA}$ termination signal in the following special case:

The GPCM is the subsystem of the memory controller that supports provision of chip-select signals ($\overline{CS}$*x*) for slaves that provide their TA signal external to the MPC855T (OR*x*[SETA] = 1). However, the GPCM keeps its chip-select asserted only until the first $\overline{TA}$ is sampled.

The GPCM cannot be used to burst to an external device the requires that the chip-select signal remain asserted throughout a burst transaction. However, if the device requires only that the chip-select be asserted up to the first data beat of the burst, it is possible to burst to this device. The user can program OR*x*[SETA] = 1 and OR*x*[BIH] = 0 to enable this operation. This is the only case in which it is valid to program OR*x*[BIH] = 0 for a chip-select controlled by the GPCM.

During a burst cycle, the user sees the chip-select assertion follow the same pattern as for a single-beat cycle. However, $\overline{BI}$ remains negated, and the burst continues for the following data beats after the negation of chip-select following $\overline{TA}$ for the first data beat.

Note also the following:

- Address incrementing is not provided in this mode. Addresses driven by the MPC855T remain the same throughout the cycle.
- The external slave must provide $\overline{TA}$ for all beats of the burst.

15.6 User-Programmable Machines (UPMs)

The two user-programmable machines (UPMs) are flexible interfaces that connect to a wide range of memory devices. At the heart of each UPM is an internal-memory RAM array that specifies the logical value driven on the external memory controller pins 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 external bus clock period on the byte-select and chip-select lines. Figure 15-31 shows the basic operation of each UPM. The following events initiate a UPM cycle:

- Any internal or external master requests an external memory access to an address space mapped to a chip-select serviced by the UPM
- A UPM periodic timer expires and requests a transaction, such as a DRAM refresh
- A transfer error or reset generates an exception request
- The MCR receives a RUN command from the CPU

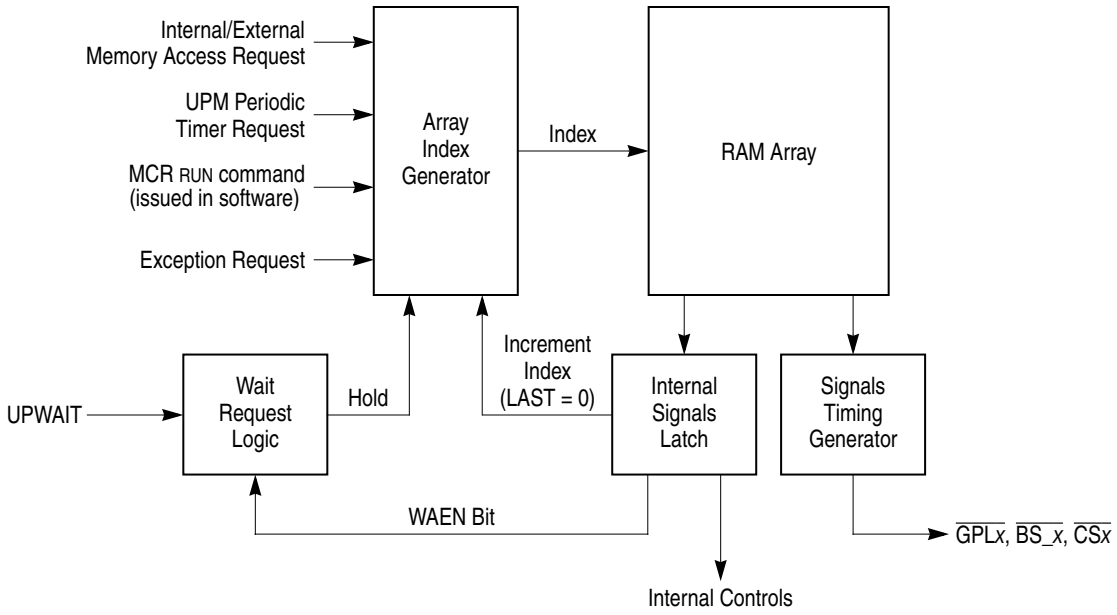


Figure 15-31. User-Programmable Machine Block Diagram

The RAM array contains 64 32-bit RAM words. The signal timing generator loads the RAM word from the RAM array to drive the general-purpose lines, byte-selects, and chip-selects. If the UPM reads a RAM word with WAEN set, the external UPWAIT signal is sampled and synchronized by the memory controller and the current request is frozen (if and while UPWAIT remains asserted).

15.6.1 Requests

An internal or external master's request for a memory access initiates one of the following patterns:

- Read single-beat pattern (RSS)
- Read burst cycle pattern (RBS)
- Write single-beat pattern (WSS)
- Write burst cycle pattern (WBS)

These patterns are described in Section 15.6.1.1, "Internal/External Memory Access Requests."

A UPM periodic timer request pattern initiates a periodic timer pattern (PTS), as described in Section 15.6.1.2, "UPM Periodic Timer Requests."

An exception (reset or machine check triggered by the assertion of $\overline{\text{TEA}}$) occurring while another UPM pattern is running initiates an exception condition pattern (EXS).

A special pattern in the RAM array is associated with each of these cycle types. Figure 15-32 shows the start addresses of these patterns in the UPM RAM, according to cycle type. MCR-initiated RUN commands, however, can initiate patterns starting at any of the 64 UPM RAM words.

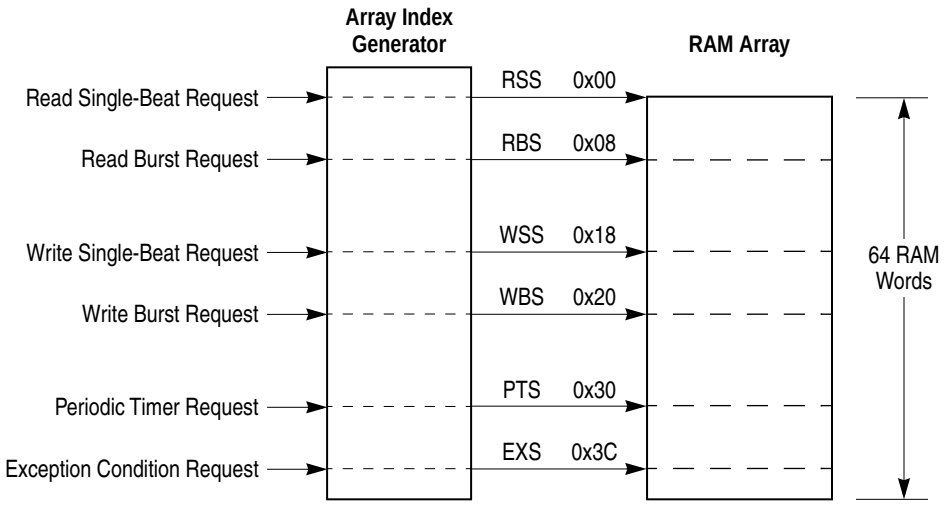


Figure 15-32. RAM Array Indexing

15.6.1.1 Internal/External Memory Access Requests

When an internal master requests a new access to external memory, the address and type of transfer are compared to each valid bank defined in BR*x*. The value in BR*x*[MS] selects the UPM to handle the memory access. The user must ensure that the UPM is appropriately initialized before a request.

External memory access requests are single-beat and burst reads and writes. A single-beat transfer transfers one operand consisting of a single byte, half word, or word. A burst transfer transfers four words. A single-beat cycle starts with one transfer start and ends with one transfer acknowledge. For 32-bit accesses, the burst cycle starts with one transfer start but ends after four transfer acknowledges. A 16-bit bus requires 8 transfer acknowledges; an 8-bit bus requires 16.

15.6.1.2 UPM Periodic Timer Requests

Each UPM contains a periodic timer that can be programmed to generate periodic service requests of a particular pattern in the RAM array. Figure 15-33 shows the hardware associated with memory periodic timer request generation. In general, the periodic timer is used for refresh cycle operation. M*x*MR[PT*x*] defines the period for the timers associated with UPM*x*. If M*x*MR[PT*x*E] is set, the periodic timer of UPM*x* requests a transaction when the timer period expires.

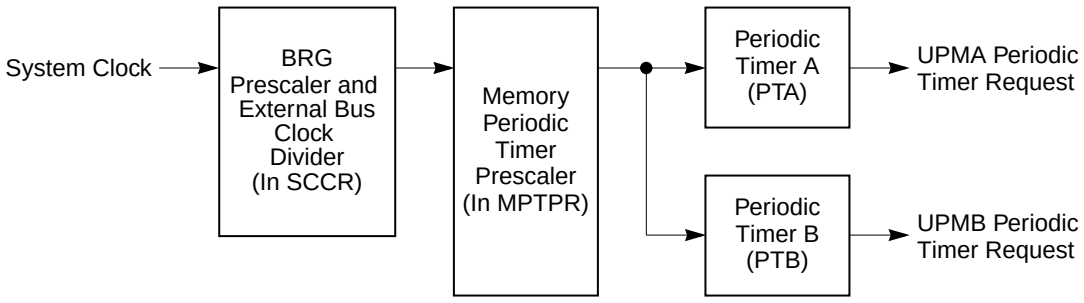


Figure 15-33. Memory Periodic Timer Request Block Diagram

15.6.1.3 Software Requests—MCR RUN Command

Software can start a request to the UPM by issuing a RUN command to the MCR. Some memory devices have their own signal handshaking protocol to put them into special modes, such as self-refresh mode. Other memory devices must be issued special commands on their control signals, such as for SDRAM initialization.

For these special cycles, the user creates a special RAM pattern that can be stored in any unused areas in the UPM RAM. Then the MCR RUN command is used to run the cycle. The UPM runs the pattern beginning at the specified RAM location until it encounters a RAM word with its LAST bit set.

15.6.1.4 Exception Requests

When the MPC855T under UPM control initiates an access to a memory device, the external device may assert $\overline{\text{TEA}}$, $\overline{\text{SRESET}}$, or $\overline{\text{HRESET}}$. The UPM provides a mechanism by which memory control signals can meet the timing requirements of the device without losing data. The mechanism is the exception pattern which defines how the UPM deasserts its signals in a controlled manner.

15.6.2 Programming the UPM

The UPM is a microsequencer that requires microinstructions or RAM words to generate signal timings for different memory cycles. Program the UPMs in the following steps:

1. Write patterns into the RAM array.
2. Program MPTPR.
3. Program the machine mode register (MAMR and MBMR).
4. Set up BR*x* and OR*x*.

15.6.3 Control Signal Generation Timing

Fields in the RAM word specify the value of the various external signals at each clock edge. The signal timing generator causes external signals to behave according to the timing specified in the current RAM word. Figure 15-34 and Figure 15-35 show the clock schemes of the UPMs in the memory controller. The clock phases shown reflect timing windows during which generated signals can change state. Figure 15-34 shows the clock scheme selected when the SCCR[EBDF] = 00; CLKOUT is the same as system clock.

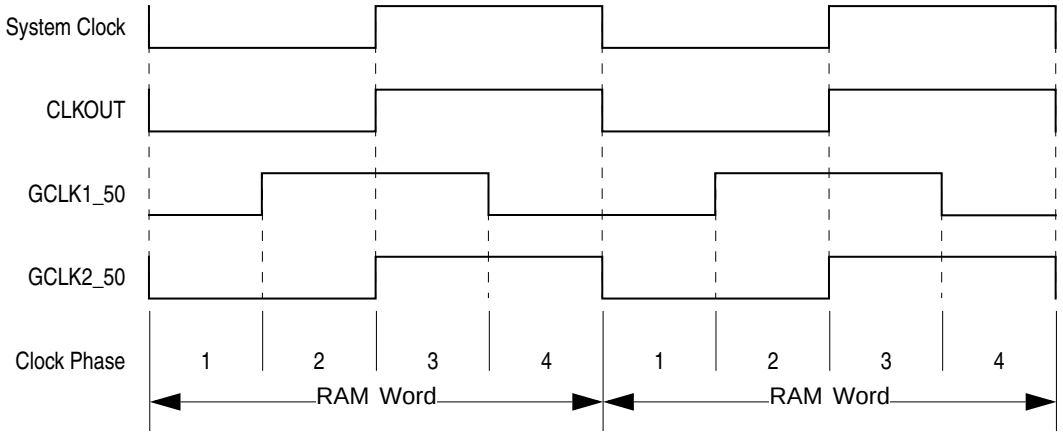


Figure 15-34. UPM Clock Scheme One (Division Factor = 1)

In Figure 15-35, if SCCR[EBDF] = 01, CLKOUT equals the system clock divided by 2. In this scheme GCLK1_50 does not have a 50% duty cycle.

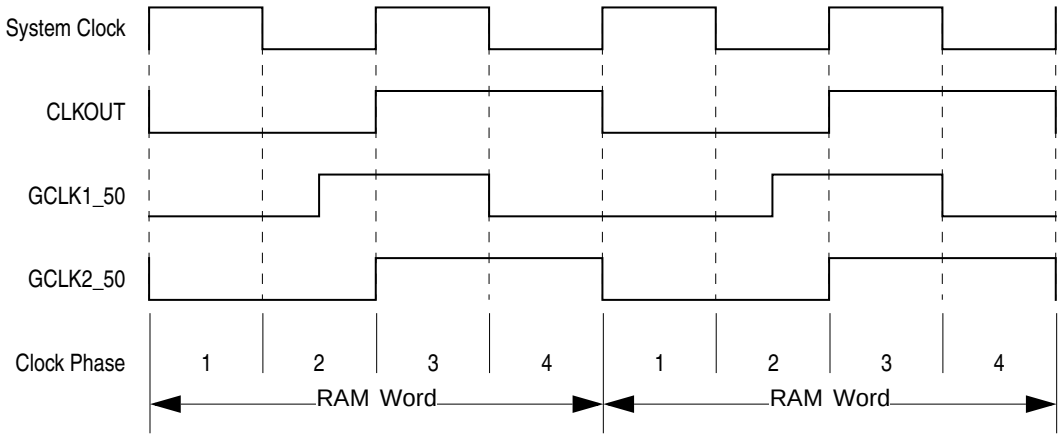


Figure 15-35. UPM Clock Scheme Two (Division Factor = 2)

The state of the external signals may change (if specified in the RAM array) at any edge of GCLK1_50 and GCLK2_50, plus a propagation delay, specified in the *MPC855T Hardware Specifications*. Note however that only the $\overline{\text{CS}}$ signal corresponding to the currently accessed bank will be manipulated by the UPM pattern when it runs. The $\overline{\text{BS}}$ signal assertion and negation timing is also specified for each cycle in the RAM word, but which of the four $\overline{\text{BS}}$ signals will be manipulated by the being run depends on the port size of the specified bank, the external address accessed, and the value of TSIZ*n*. The $\overline{\text{GPL}}$ lines toggle as programmed for any access that initiates a particular pattern, but resolution of control is slightly more limited.

The examples in Figure 15-36 and Figure 15-37 show how to control the timing of $\overline{\text{CS}}$, $\overline{\text{GPL1}}$, and $\overline{\text{GPL2}}$. UPM RAM words determine the values of the CST[1–4], G1T3, G1T4, G2T3, and G2T4 bits, which specify the timing of chip-selects, byte-selects, and GPL signals based on the edges of GCLK1_50 or GCLK2_50. The clock phases shown refer to the timing windows when the signals controlled by these bits in the RAM word are driven.

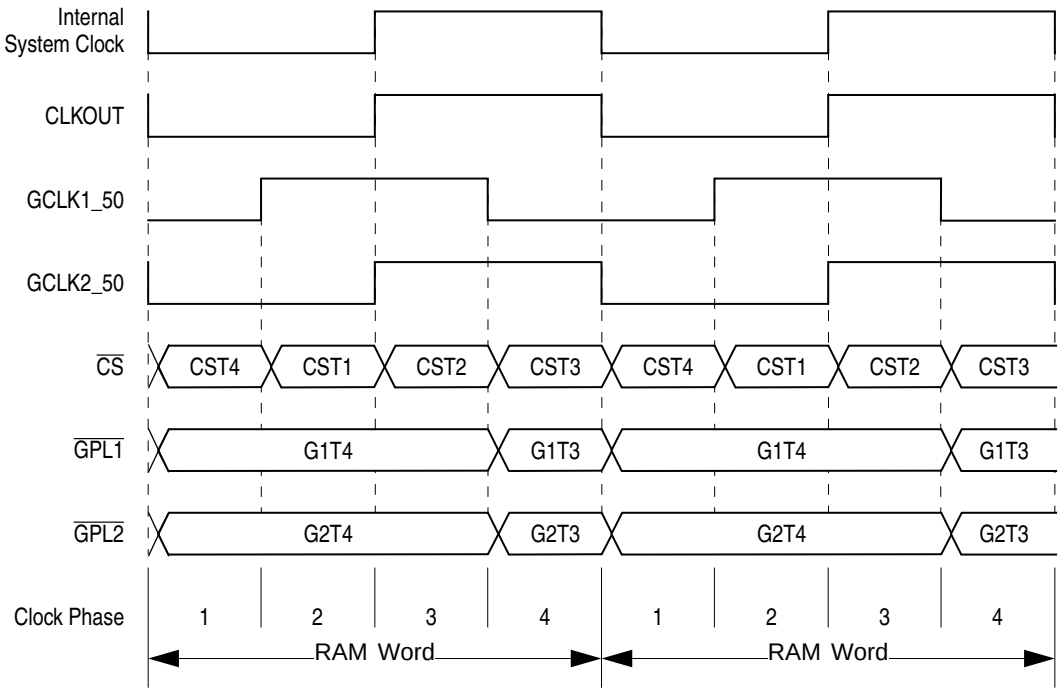


Figure 15-36. UPM Signals Timing Example One (Division Factor = 1, EBDF = 00)

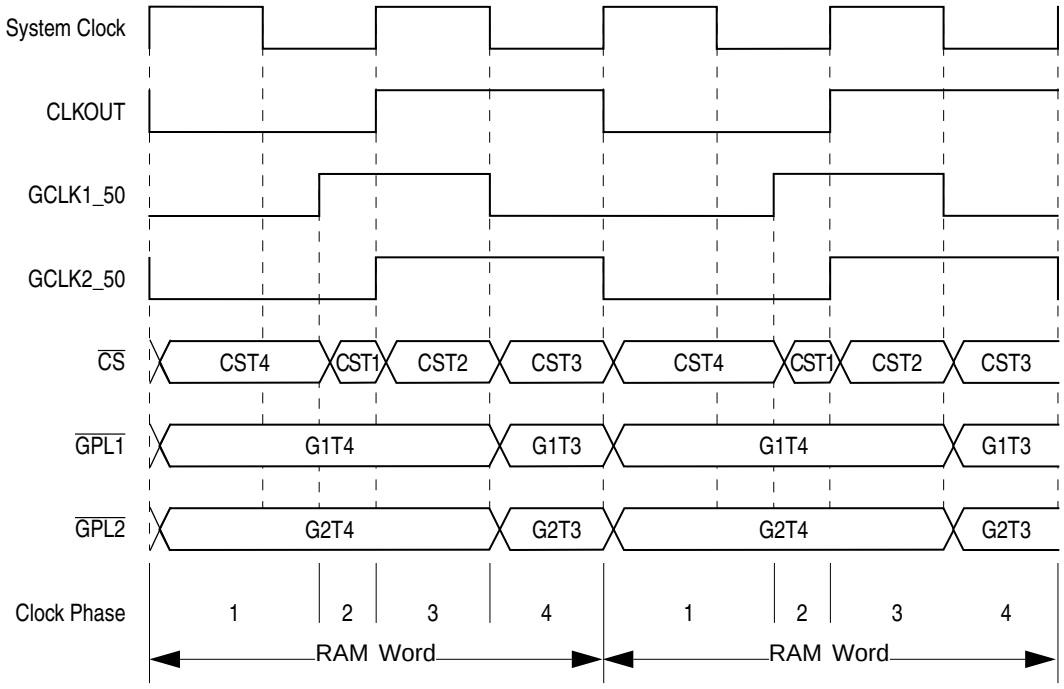


Figure 15-37. UPM Signals Timing Example Two (Division Factor = 2, EBDF = 01)

15.6.4 The RAM Array

The RAM array for each UPM is 64 locations deep and 32 bits wide, as shown in Figure 15-38. The signals at the bottom of Figure 15-38 are UPM outputs. The selected $\overline{CS}$ is for the bank that matches the current address. The selected $\overline{BS}$ is for the byte lanes read or written by the access.

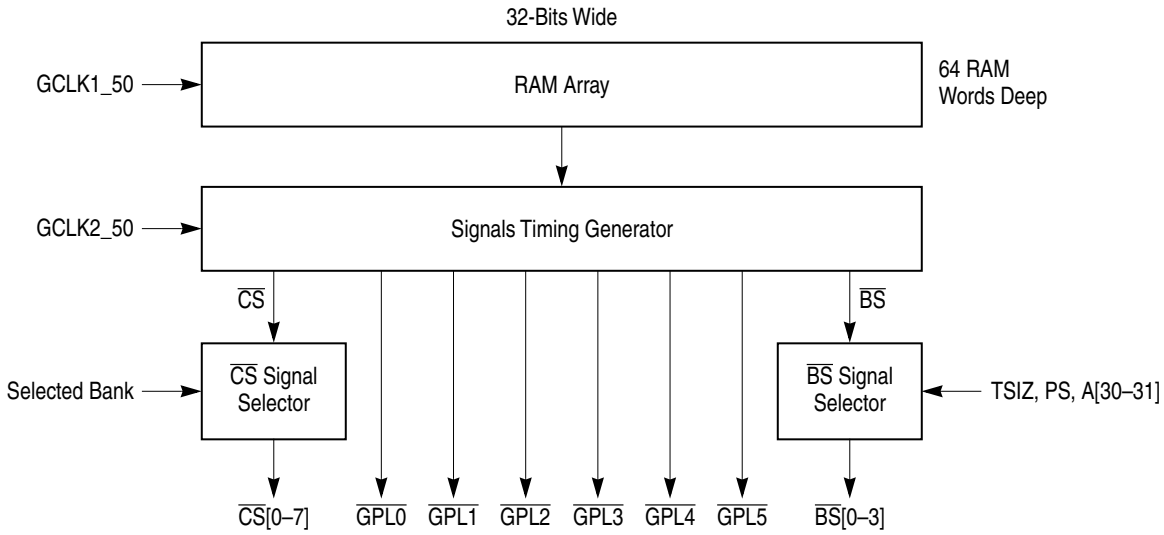


Figure 15-38. RAM Array and Signal Generation

Each UPM request (except software requests issued via RUN commands in MCR) has a special address that specifies the beginning of the associated pattern in the UPM RAM array. Table 15-13 shows start addresses of the UPM RAM words for each request type. (See also Figure 15-32.)

Table 15-13. UPM Start Address Locations

Request to Be Serviced	UPM Start Address
Read single beat cycle (RSS)	0x00
Read burst cycle (RBS)	0x08
Write single beat cycle (WSS)	0x18
Write burst cycle (WBS)	0x20
Periodic timer request (PTS)	0x30
Exception (EXS)	0x3C

15.6.4.1 RAM Words

The RAM word, shown in Figure 15-39, is a 32-bit microinstruction stored in one of 64 locations in the RAM array. It specifies timing for external signals controlled by the UPM.

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															
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	—		LOOP	EXEN	AMX		NA	UTA	TODT	LAST
Reset	—															
R/W	R/W															
Addr	(All 32 bits of the RAM word are addressed as shown in the address row above.)															

Figure 15-39. The RAM Word

Table 15-14 describes RAM word fields.

Table 15-14. RAM Word Bit Settings

Bit	Name	Description
0	CST4	Chip-select timing 4. Defines the state of $\overline{CS}$ during clock phase 1. 0 Asserted at the falling edge of GCLK2_50. 1 Negated at the falling edge of GCLK2_50.
1	CST1	Chip-select timing 1. Defines the state of $\overline{CS}$ during clock phase 2. 0 Asserted at the rising edge of GCLK1_50. 1 Negated at the rising edge of GCLK1_50.
2	CST2	Chip-select timing 2. Defines the state of $\overline{CS}$ during clock phase 3. 0 Asserted at the rising edge of GCLK2_50. 1 Negated at the rising edge of GCLK2_50.
3	CST3	Chip-select timing3. Defines the state of $\overline{CS}$ during clock phase 4. 0 Asserted at the falling edge of GCLK1_50. 1 Negated at the falling edge of GCLK1_50.
4	BST4	Byte-select timing 4. Defines the state of $\overline{BS}$ during clock phase 1. 0 Asserted at the falling edge of GCLK2_50. 1 Negated at the falling edge of GCLK2_50. The final value of the $\overline{BS}$ lines depends on the values of BRx[PS], the TSIZ lines, and A[30–31] for the access. See **Section 15.6.4.3, "Byte-Select Signals (BSTx)**."
5	BST1	Byte-select timing 1. Defines the state of $\overline{BS}$ during clock phase 2. 0 Asserted at the rising edge of GCLK1_50. 1 Negated at the rising edge of GCLK1_50. The final value of the $\overline{BS}$ lines depends on the values of BRx[PS], the TSIZ lines, and A[30–31] for the access. See **Section 15.6.4.3, "Byte-Select Signals (BSTx)**."
6	BST2	Byte-select timing 2. Defines the state of $\overline{BS}$ during clock phase 3. 0 Asserted at the rising edge of GCLK2_50. 1 Negated at the rising edge of GCLK2_50 The final value of the $\overline{BS}$ lines depends on the values of BRx[PS], the TSIZ lines, and A[30–31] for the access. See **Section 15.6.4.3, "Byte-Select Signals (BSTx)**."

Table 15-14. RAM Word Bit Settings (continued)

Bit	Name	Description
7	BST3	Byte-select timing 3. Defines the state of $\overline{BS}$ during clock phase 4. 0 Asserted at the falling edge of GCLK1_50. 1 Negated at the falling edge of GCLK1_50. The final value of the $\overline{BS}$ lines depends on the values of BRx[PS], the TSIZ lines, and A[30–31] for the access. See **Section 15.6.4.3, "Byte-Select Signals (BSTx)**."
8–9	G0L	General-purpose line 0 lower. Defines the state of $\overline{GPL0}$ during phases 1–3. 10 Asserted at the falling edge of GCLK2_50. 11 Negated at the falling edge of GCLK2_50. 00 Driven at the falling edge of GCLK2_50 with an address signal as defined in M*x*MR[G0CL*x*].
10–11	G0H	General-purpose line 0 higher. Defines the state of $\overline{GPL0}$ during phase 4. 10 Asserted at the falling edge of GCLK1_50. 11 Negated at the falling edge of GCLK1_50. 00 Driven at the falling edge of GCLK1_50 with an address signal as defined in M*x*MR[G0CL*x*].
12	G1T4	General-purpose line 1 timing 4. Defines the state of $\overline{GPL1}$ during phase 1–3. 0 Asserted at the falling edge of GCLK2_50. 1 Negated at the falling edge of GCLK2_50.
13	G1T3	General-purpose line 1 timing 3. Defines the state of $\overline{GPL1}$ during phase 4. 0 Asserted at the falling edge of GCLK1_50. 1 Negated at the falling edge of GCLK1_50.
14	G2T4	General-purpose line 2 timing 4. Defines the state of $\overline{GPL2}$ during phase 1–3. 0 Asserted at the falling edge of GCLK2_50. 1 Negated at the falling edge of GCLK2_50.
15	G2T3	General-purpose line 2 timing 3. Defines the state of $\overline{GPL2}$ during phase 4. 0 Asserted at the falling edge of GCLK1_50. 1 Negated at the falling edge of GCLK1_50.
16	G3T4	General-purpose line 3 timing 4. Defines the state of $\overline{GPL3}$ during phase 1–3. 0 Asserted at the falling edge of GCLK2_50. 1 Negated at the falling edge of GCLK2_50.
17	G3T3	General-purpose line 3 timing 3. Defines the state of $\overline{GPL3}$ during phase 4. 0 Asserted at the falling edge of GCLK1_50. 1 Negated at the falling edge of GCLK1_50.
18	G4T4/ DLT3	General-purpose line 4 timing 4/delay time 3. The function is determined by M*x*MR[GPL*x*4DIS].
	G4T4	If M*x*MR defines UPWAITx/$\overline{GPL\_x4}$ as an output ($\overline{GPL\_x4}$), this bit functions as G4T4: 0 The value of $\overline{GPL4}$ at the falling edge of GCLK2_50 will be 0. 1 The value of $\overline{GPL4}$ at the falling edge of GCLK2_50 will be 1.
	DLT3	If M*x*MR[GPL*x*4DIS] = 1, UPWAITx is chosen and this bit functions as DLT3. 0 The data bus should be sampled at the rising edge of GCLK2_50 for a read in this cycle. 1 The data bus should be sampled at the falling edge of GCLK2_50 for a read in this cycle.

Table 15-14. RAM Word Bit Settings (continued)

Bit	Name	Description
19	G4T3/WAEN	General-purpose line 4 timing 3/wait enable. Function depends on the value of M*x*MR[GPL*x*4DIS].
	G4T3	If M*x*MR[GPL*x*4DIS] = 0, G4T3 is selected. 0 The value of $\overline{\text{GPL4}}$ at the falling edge of GCLK1_50 will be 0. 1 The value of $\overline{\text{GPL4}}$ at the falling edge of GCLK1_50 will be 1.
	WAEN	If M*x*MR[GPL*x*4DIS] = 1, WAEN is selected. 0 The UPWAITx function is disabled. 1 The logical value of the UPM-controlled external signals are frozen when UPWAITx is asserted. UPWAITx is sampled on the falling edge of GCLK2_50. See Figure 15-45. for more information.
20	G5T4	General-purpose line 5 timing 4. Defines the state of $\overline{\text{GPL5}}$ during phase 1–3. 0 The value of $\overline{\text{GPL5}}$ at the falling edge of GCLK2_50 will be 0. 1 The value of $\overline{\text{GPL5}}$ at the falling edge of GCLK2_50 will be 1.
21	G5T3	General-purpose line 5 timing 3. Defines the state of $\overline{\text{GPL5}}$ during phase 4. 0 The value of $\overline{\text{GPL5}}$ at the falling edge of GCLK1_50 will be 0. 1 The value of $\overline{\text{GPL5}}$ at the falling edge of GCLK1_50 will be 1.
22–23	—	Reserved, should be cleared.
24	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 the loop end word. RAM words between the start and end are defined as the loop. The number of times the UPM executes this loop is defined in the corresponding loop field of the M*x*MR. 0 The current RAM word is not the loop start word or loop end word. 1 The current RAM word is the start or end of a loop. See Section 15.6.4.5, “Loop Control (LOOP).”
25	EXEN	Exception enable. If an external device asserts $\overline{\text{TEA}}$ or $\overline{\text{RESET}}$, EXEN allows branching to an exception pattern at the exception start address (EXS) 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 the exception pattern 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 $\overline{\text{SRESET}}$.
26–27	AMX	Address multiplexing. Determines the source of A[0–31] at the falling edge of GCLK1_50. 00 A[0–31] is the non-multiplexed address. For example, column address. 01 Reserved. 10 A[0–31] is the address requested by the internal master multiplexed according to M*x*MR[AM*x*]. For example, row address. 11 A[0–31] is the contents of MAR. Used for example, during SDRAM mode initialization.
28	NA	Next address. Determines when the address is incremented during a burst access. 0 The address increment function is disabled 1 The address is incremented in the next cycle. In conjunction with the BR*x*[PS], the increment value of A[28–31] and/or BADDR[28–30] at the falling edge of GCLK1_50 is as follows If the accessed bank has a 32-bit port size, the value is incremented by 4. If the accessed bank has a 16-bit port size, the value is incremented by 2. If the accessed bank has an 8-bit port size, the value is incremented by 1. Note: The value of NA is relevant only when the UPM serves a burst-read or burst-write request. NA is reserved under other patterns.

Table 15-14. RAM Word Bit Settings (continued)

Bit	Name	Description
29	UTA	UPM transfer acknowledge. Controls the state of $\overline{TA}$ sampled by the external bus interface in the current memory cycle. $\overline{TA}$ is output at the rising edge of GCLK2_50. 0 $\overline{TA}$ is driven low on the rising edge of GCLK2_50. The bus master samples it low in the next clock cycle. 1 $\overline{TA}$ is driven high on the rising edge of GCLK2_50.
30	TODT	Turn-on disable timer. Controls the disable timer mechanism. This bit has meaning only in RAM words for which UTA = 0; otherwise it is a don't care. 0 The disable timer is turned off. 1 The disable timer for the current bank is activated preventing a new access to the same bank (when controlled by the UPMs) until the disable timer expires. For example, precharge time.
31	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.

15.6.4.2 Chip-Select Signals (CST*x*)

If BR*x*[MS] of the accessed bank selects a UPM on the currently requested cycle the UPM manipulates the $\overline{CS}$ signal for that bank with timing as specified in the UPM RAM word. The selected UPM affects only assertion and negation of the appropriate $\overline{CSx}$ signal. The state of the selected $\overline{CSx}$ signal of the corresponding bank depends on the value of each CST*n* bit.

Figure 15-40 and the timing diagrams in Figure 15-36 and Figure 15-37 shows how UPMs control $\overline{CS}$ signals.

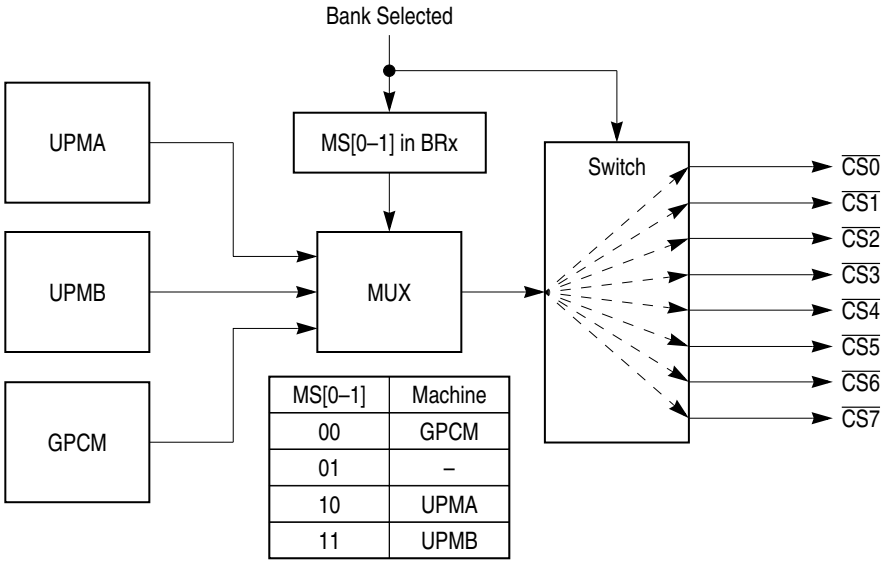


Figure 15-40. $\overline{CSx}$ Signal Selection

15.6.4.3 Byte-Select Signals (BST*x*)

BR*x*[MS] of the accessed memory bank selects a UPM on the currently requested cycle. The selected UPM affects only the assertion and negation of the appropriate $\overline{BS}$ signal; its timing as specified in the RAM word. The state of each $\overline{BS[0–3]}$ signal depends on the value of each BST*x* bit and the values of BR*x*[PS], TSIZ*n*, and A[30–31] in the current cycle.The $\overline{BS}$ signals are also controlled by the port size of the accessed bank, the transfer size of the transaction, and the address accessed. Figure 15-41 shows how UPMs control $\overline{BS}$ signals.

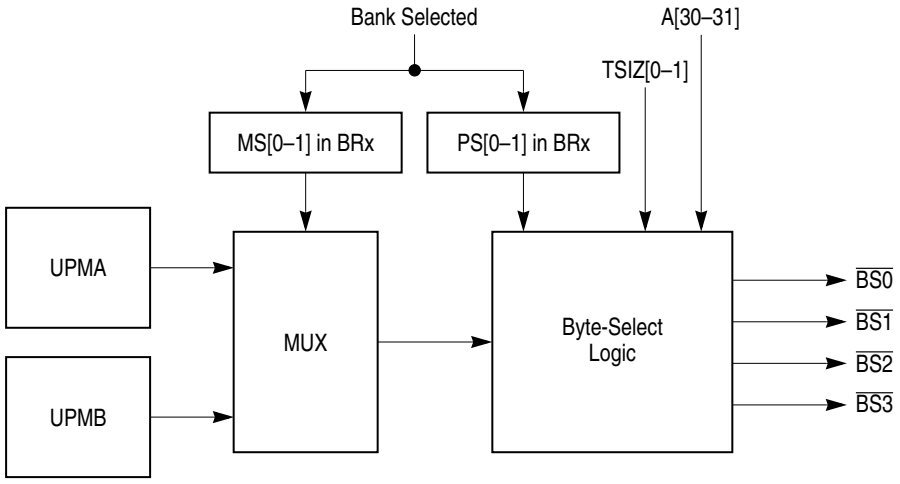


Figure 15-41. $\overline{BS}$*x* Signal Selection

The uppermost byte select ($\overline{BS0}$) indicates that D[0–7] contains valid data during a cycle. Likewise, $\overline{BS1}$ indicates that D[8–15] contains valid data, $\overline{BS2}$ indicates that D[16–23] contains valid data, and $\overline{BS3}$ indicates that D[24–31] contains valid data during a cycle. Table 15-15 shows how $\overline{BS}$ signals affect 32-, 16-, and 8-bit accesses. Note that for a periodic timer request and a memory command request, the $\overline{BS}$ signals are determined only by the port size of the bank.

Table 15-15. Enabling Byte-Selects

Transfer Size	TSIZ		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			

15.6.4.4 General-Purpose Signals (G*x*T*x*, G0*x*)

The general-purpose signals (GPL[1–5]) have two bits in the RAM word that define the logical value of the signal to be changed at the falling edge of GCLK1_50 or GCLK2_50. GPL0 has two 2-bit fields that perform this function plus an additional function explained below. GPL5 and GPL0 offer the following enhancements beyond the other GPL**x** signals:

- GPL5 can be controlled during phase 4 of the first clock cycle according to the value of G5LS, as shown in Figure 15-42. This allows it to assert earlier (simultaneous with TS, for an internal master), which can speed up the memory interface, particularly when GPL5 is used as a control signal for external address multiplexers.

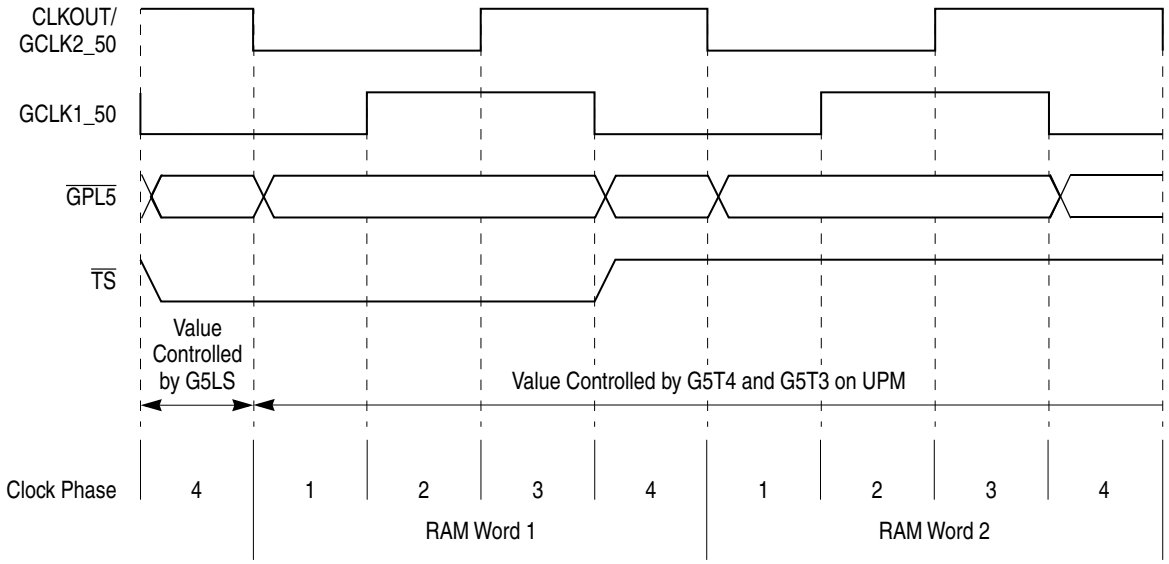


Figure 15-42. Early GPL5 Control

- GPL0 can be controlled by an address line specified in M*x*MR[G0CL*x*]. To use this feature, set G0H and G0L in the RAM word. For example, for a SIMM with multiple banks, this address line can be used to switch between banks.

The state of GPL_x5 logic depends on the defined in Table 15-16. In the first clock cycle of the slave access, GPL_x5 reflects the value of OR*x*[G5LS]; in subsequent cycles, its state is determined by G5T4 and G5T3 in the RAM word. If the UPMB controls slave access, OR*x*[G5LA] can be used to select the active GPL_x5 signal. G5LS applies only to memory requests and not to RAM words executed by the RUN command, exception, or memory periodic timer requests.

Table 15-16. $\overline{\text{GPL\_X5}}$ Signal Behavior

Controlling Machine		ORx		RAM Word		$\overline{\text{GPL\_X5}}$ Behavior at the Controlling Clock Edge
Memory Access	Slave Access Clock Cycle	G5LA	G5LS	G5T4	G5T3	
GPCM	x	N/A	N/A	x	x	$\overline{\text{GPL\_A5}}$ and $\overline{\text{GPL\_B5}}$ do not change their value.
UPMA	First	x	0	x	x	$\overline{\text{GPL\_A5}}$ is driven low at the falling edge of GCLK1_50.
			1			$\overline{\text{GPL\_A5}}$ is driven high at the falling edge of GCLK1_50.
	Second, third...	x	x	0	x	$\overline{\text{GPL\_A5}}$ is driven low at the falling edge of GCLK2_50 in the current UPM cycle.
				1	x	$\overline{\text{GPL\_A5}}$ is driven high at the falling edge of GCLK2_50 in the current UPM cycle.
				x	0	$\overline{\text{GPL\_A5}}$ is driven low at the falling edge of GCLK1_50 in the current UPM cycle.
				x	1	$\overline{\text{GPL\_A5}}$ is driven high at the falling edge of GCLK1_50 in the current UPM cycle.
UPMB	First	0	0	x	x	$\overline{\text{GPL\_B5}}$ is driven low at the falling edge of GCLK1_50.
			1			$\overline{\text{GPL\_B5}}$ is driven high at the falling edge of GCLK1_50.
		1	0	x	x	$\overline{\text{GPL\_A5}}$ is driven low at the falling edge of GCLK1_50.
			1			$\overline{\text{GPL\_A5}}$ is driven high at the falling edge of GCLK1_50.
	Second, third...	0	x	0	x	$\overline{\text{GPL\_B5}}$ is driven low at the falling edge of GCLK2_50 in the current UPM cycle.
				1	x	$\overline{\text{GPL\_B5}}$ is driven high at the falling edge of GCLK2_50 in the current UPM cycle.
				x	0	$\overline{\text{GPL\_B5}}$ is driven low at the falling edge of GCLK1_50 in the current UPM cycle.
				x	1	$\overline{\text{GPL\_B5}}$ is driven high at the falling edge of GCLK1_50 in the current UPM cycle.
		1	x	0	x	$\overline{\text{GPL\_A5}}$ is driven low at the falling edge of GCLK2_50 in the current UPM cycle.
				1	x	$\overline{\text{GPL\_A5}}$ is driven high at the falling edge of GCLK2_50 in the current UPM cycle.
				x	0	$\overline{\text{GPL\_A5}}$ is driven low at the falling edge of GCLK1_50 in the current UPM cycle.
				x	1	$\overline{\text{GPL\_A5}}$ is driven high at the falling edge of GCLK1_50 in the current UPM cycle.

15.6.4.5 Loop Control (LOOP)

The LOOP bit in the RAM word (bit 24) specifies the beginning and end of a set of UPM RAM words that are to be repeated. The first time LOOP = 1, the memory controller recognizes it as a loop start word and loads the memory loop counter with the corresponding contents of the loop field shown in Table 15-17. The next RAM word for

which LOOP = 1 is recognized as a loop end word. When it is reached, the loop counter is decremented by one.

Continued loop execution depends on the loop counter. If the counter is not zero, the next RAM word executed is the loop start word. Otherwise, the next RAM word executed is the one after the loop end word. Loops can be executed sequentially but cannot be nested.

Table 15-17. M*x*MR Loop Field Usage

Request Serviced	Loop Field
Read single-beat cycle	RLFx
Read burst cycle	RLFx
Write single-beat cycle	WLFx
Write burst cycle	WLFx
Periodic timer expired	TLFx

15.6.4.6 Exception Pattern Entry (EXEN)

When the MPC855T under UPM control begins accessing a memory device, the external device may assert $\overline{\text{TEA}}$, $\overline{\text{SRESET}}$, or $\overline{\text{HRESET}}$. An exception occurs when one of these signals is asserted by an external device and the MPC855T begins closing the memory cycle transfer. When one of these exceptions is recognized and EXEN in the RAM word is set, the UPM branches to the special exception start address (EXS) and begins operating as the pattern defined there specifies. See Table 15-15. The user should provide an exception pattern to deassert signals controlled by the UPM in a controlled fashion. For DRAM control, a handler should negate $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ to prevent data corruption. If EXEN = 0, exceptions are deferred and execution continues. After the UPM branches to the exception start address, it continues reading until the LAST bit is set in the RAM word.

15.6.4.7 Address Multiplexing (AMX)

To support many devices with multiplexed address signals, the upper address signals can be driven on the lower address lines. M*x*MR[AMA] and M*x*MR[AMB] control which upper address signals are on which lower address signals.

Note that this feature of internally multiplexing address signals should only be used in a system where the MPC855T is the only external bus master. If other devices can be bus masters, address multiplexing must be done in external logic. One of the UPM's output signals can be used to control this external multiplexing logic; $\overline{\text{GPL5}}$ has been specifically enhanced for this. See the description of $\overline{\text{GPL5}}$ in Section 15.6.4.4, "General-Purpose Signals (GxTx, G0x)."

OR*x*[SAM] and the AMX field of the RAM words determine when the multiplexing occurs. OR*x*[SAM] controls address multiplexing for the first clock cycle. The AMX field in the RAM word determines the multiplexing for subsequent clock cycles. As an address is

driven off of the falling edge of GCLK1–50, the address in a particular clock cycle is actually controlled by the previous RAM word, as shown in Figure 15-43.

The AMX field can be used to output the contents of MAR on the address signals. Figure 15-43 shows address multiplex timing.

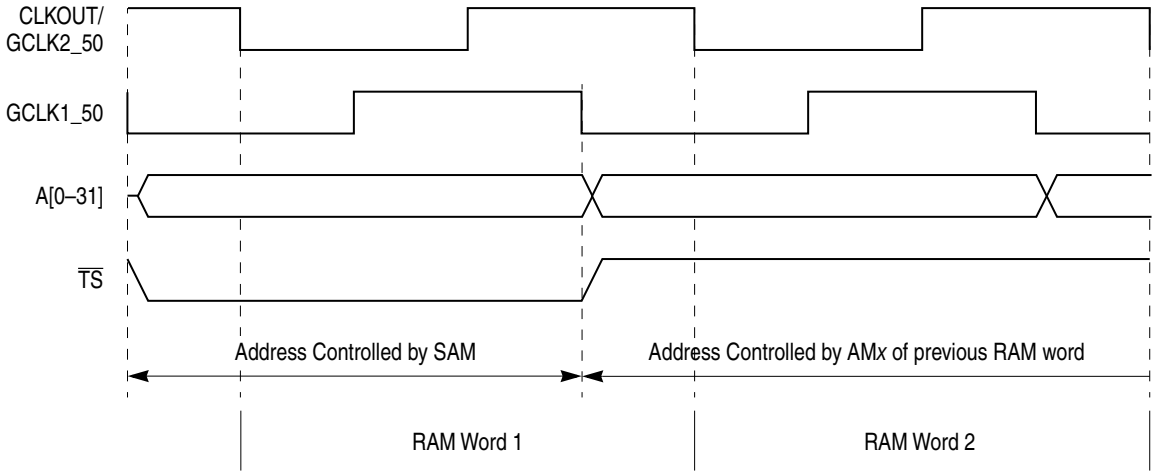


Figure 15-43. Address Multiplex Timing

Table 15-18 shows how M*x*MR[AM*x*] settings affect address multiplexing.

Table 15-18. Address Multiplexing

AM*x*	External Bus Address Pin	A16	A17	A18	A19	A20	A21	A22	A23	A24	A25	A26	A27	A28	A29	A30	A31
000	Signal Driven on External Pin when Address Multiplexing is Enabled	—	—	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20	A21	A22	A23
001		—	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		—	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		—	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18

Table 15-19 shows how AM*x* can be defined to interface with a range of DRAM modules.

Table 15-19. AMA/AMB Definition for DRAM Interface

Data Bus Width	Memory Size	DRAM Address Pin Number		MPC855T Address Pin Connection	AM*x*
		Row	Column		
8 bits	64 Kbyte	8	8	A24–A31	000
	128 Kbyte	9		A23–A31	
	256 Kbyte	10		A22–A31	
	512 Kbyte	11		A21–A31	
	1 Mbyte	12		A20–A31	
	2 Mbyte	13		A19–A31	
	4 Mbyte	14		A18–A31	
	256 Kbyte	9	9	A23–A31	001
	512 Kbyte	10		A22–A31	
	1 Mbyte	11		A21–A31	
	2 Mbyte	12		A20–A31	
	4 Mbyte	13		A19–A31	
	8 Mbyte	14		A18–A31	
	16 Mbyte	15		A17–A31	

Table 15-19. AMA/AMB Definition for DRAM Interface (continued)

Data Bus Width	Memory Size	DRAM Address Pin Number		MPC855T Address Pin Connection	AM*x*
		Row	Column		
8 bits	1 Mbyte	10	10	A22–A31	010
	2 Mbyte	11		A21–A31	
	4 Mbyte	12		A20–A31	
	8 Mbyte	13		A19–A31	
	16 Mbyte	14		A18–A31	
	32 Mbyte	15		A17–A31	
	64 Mbyte	16		A16–A31	
	4 Mbyte	11	11	A21–A31	011
	8 Mbyte	12		A20–A31	
	16 Mbyte	13		A19–A31	
	32 Mbyte	14		A18–A31	
	64 Mbyte	15		A17–A31	
	16 Mbyte	12	12	A20–A31	100
	32 Mbyte	13		A19–A31	
	64 Mbyte	14		A18–A31	
	128 Mbyte	15		A17–A31	
	256 Mbyte	16		A16–A31	
	64 Mbyte	13	13	A19–A31	101
	128 Mbyte	14		A18–A31	
	256 Mbyte	15		A17–A31	

Table 15-19. AMA/AMB Definition for DRAM Interface (continued)

Data Bus Width	Memory Size	DRAM Address Pin Number		MPC855T Address Pin Connection	AM*x*
		Row	Column		
16 bits	128 Kbyte	8	8	A23–A30	000
	256 Kbyte	9		A22–A30	
	512 Kbyte	10		A21–A30	
	1 Mbyte	11		A20–A30	
	2 Mbyte	12		A19–A30	
	4 Mbyte	13		A18–A30	
	512 Kbyte	9	9	A22–A30	001
	1 Mbyte	10		A21–A30	
	2 Mbyte	11		A20–A30	
	4 Mbyte	12		A19–A30	
	8 Mbyte	13		A18–A30	
	16 Mbyte	14		A17–A30	
	2 Mbyte	10	10	A21–A30	010
	4 Mbyte	11		A20–A30	
	8 Mbyte	12		A19–A30	
	16 Mbyte	13		A18–A30	
	32 Mbyte	14		A17–A30	
	64 Mbyte	15		A16–A30	
	8 Mbyte	11	11	A20–A30	011
	16 Mbyte	12		A19–A30	
	32 Mbyte	13		A18–A30	
	64 Mbyte	14		A17–A30	
	32 Mbyte	12	12	A19–A30	100
	64 Mbyte	13		A18–A30	
	128 Mbyte	14		A17–A30	
	256 Mbyte	15		A16–A30	
	128 Mbyte	13	13	A18–A30	101
	256 Mbyte	13		A17–A30	

Table 15-19. AMA/AMB Definition for DRAM Interface (continued)

Data Bus Width	Memory Size	DRAM Address Pin Number		MPC855T Address Pin Connection	AM*x*
		Row	Column		
32 bits	256 Kbyte	8	8	A22–A29	000
	512 Kbyte	9		A21–A29	
	1 Mbyte	10		A20–A29	
	2 Mbyte	11		A19–A29	
	4 Mbyte	12		A18–A29	
	1 Mbyte	9	9	A21–A29	001
	2 Mbyte	10		A20–A29	
	4 Mbyte	11		A19–A29	
	8 Mbyte	12		A18–A29	
	16 Mbyte	13		A17–A29	
	4 Mbyte	10	10	A20–A29	010
	8 Mbyte	11		A19–A29	
	16 Mbyte	12		A18–A29	
	32 Mbyte	13		A17–A29	
	64 Mbyte	14		A16–A29	
	16 Mbyte	11	11	A19–A29	011
	32 Mbyte	12		A18–A29	
	64 Mbyte	13		A17–A29	
	64 Mbyte	12	12	A18–A29	100
	128 Mbyte	13		A17–A29	
	256 Mbyte	14		A16–A29	
	256 Mbyte	13	13	A17–A29	101

15.6.4.8 Transfer Acknowledge and Data Sample Control (UTA, DLT3)

During a memory access, the UTA bit of the RAM word controls the state of $\overline{TA}$ driven by the UPM. $\overline{TA}$ is driven on the rising edge of GCLK2_50. Therefore, because $\overline{TA}$ is also sampled on the rising edge of GCLK2_50, programming UTA to assert in the RAM word causes the bus master to sample $\overline{TA}$ as asserted in the next cycle.

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 internal bus master, assuming that M*x*MR[GPL*x*4DIS] = 1.

- If G4T4/DLT3 functions as DLT3 and DLT3 = 1 in the RAM word, data is latched on the falling edge of GCLK2_50 instead of the rising edge, which is normal. This feature lets the user speed up the memory interface by latching data 1/2 clock early,

which can be useful during burst reads. This feature should be used only in systems without external synchronous bus devices.

- If G4T4/DLT3 functions as G4T4, data is latched on the rising edge of the external bus clock, as is normal in MPC855T bus operation.

Figure 15-44 shows data sampling that is controlled by the UPM.

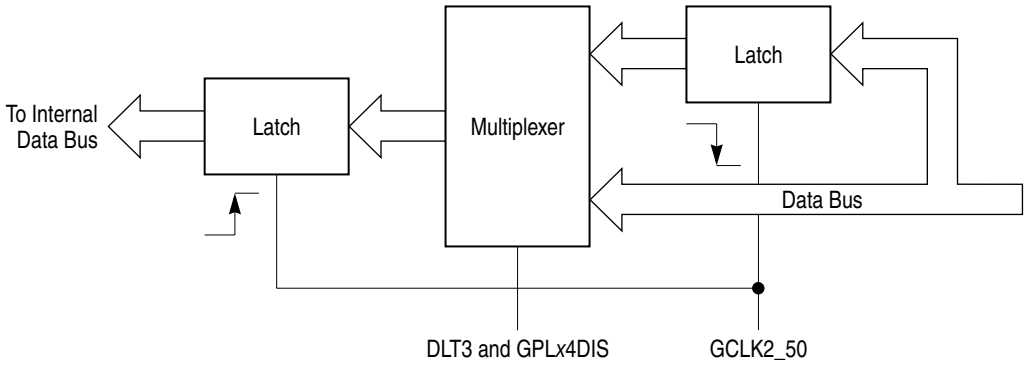


Figure 15-44. UPM Read Access Data Sampling

15.6.4.9 Disable Timer Mechanism (TODT)

The disable timer associated with each UPM allows a minimum time to be guaranteed between two successive accesses to the same memory bank. This feature is critical when DRAM requires a $\overline{\text{RAS}}$ 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 period is determined in M*x*MR[DS*x*]. The disable timer does not affect memory accesses to different banks.

TODT is usually set in the RAM word in which LAST = 1. However, it can be set in a previous RAM word, if, for example, one pattern requires *n* clocks of $\overline{\text{RAS}}$ precharge enforced outside of itself, while another pattern requires only *n* - 1.

15.6.4.10 The Last Word (LAST)

When the LAST bit is read in a RAM word, the current UPM pattern terminates and the highest priority pending UPM request (if any) is serviced immediately in the external memory transactions. If the disable timer is activated and the next access is top the same bank, the execution of the next UPM pattern is held off for the number of clock cycles specified in M*x*MR[DS*x*].

15.6.4.11 The Wait Mechanism (WAEN)

The WAEN bit can be used to enable the UPM wait mechanism in selected UPM RAM words. The wait mechanism works differently for synchronous and asynchronous masters.

15.6.4.11.1 Internal and External Synchronous Masters

If the UPM reads a RAM word with the WAEN bit set, the external UPWAIT signal is sampled and synchronized by the memory controller and the current request is frozen (if and while UPWAIT remains asserted). If the WAEN bit is set and UPWAIT was sampled high on the previous falling edge of GCLK2_50, the logical value of the external signals are frozen to the value defined at the next falling GCLK2_50 edge as programmed in the RAM word until UPWAIT is negated. This allows wait states to be inserted as required by an external device through an external signal.

Figure 15-45 shows how the WAEN bit in the word read by the UPM and the UPWAIT signal are used to hold the UPM in a particular state until UPWAIT is negated. As the example in Figure 15-45 shows, the $\overline{\text{CSx}}$ and $\overline{\text{GPL1}}$ states (C12 and F) and the WAEN value (CC) are frozen until UPWAIT is recognized as deasserted.

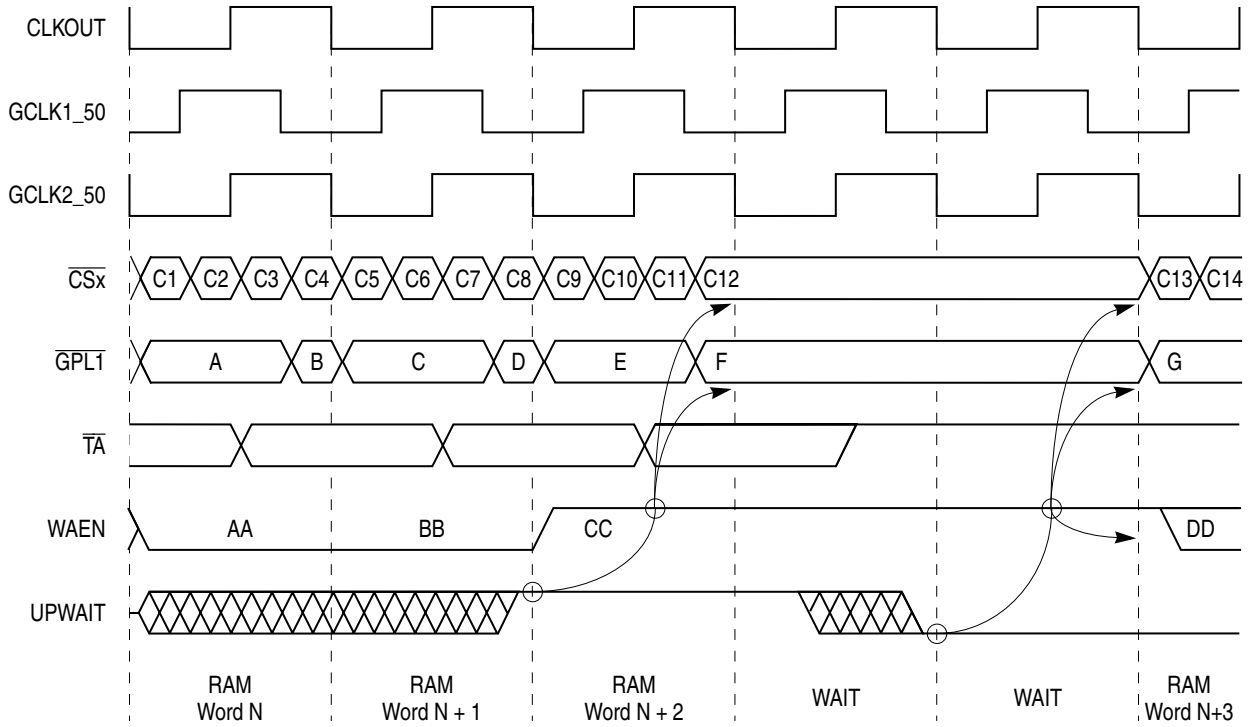


Figure 15-45. Wait Mechanism Timing for Internal and External Synchronous Masters

15.6.4.11.2 External Asynchronous Masters

For an external asynchronous master, $\overline{\text{AS}}$ is the external signal that activates the wait mechanism. The UPM enters a wait state if $\overline{\text{AS}}$ was sampled asserted on the previous falling edge of GLCK2_50 and WAEN = 1 in the current RAM word. In this wait state, external signals are frozen after the falling edge of GCLK2_50, as programmed in the RAM word in which WAEN is set. This is demonstrated in the example in Figure 15-46 in which the $\overline{\text{CSx}}$ and $\overline{\text{GPL1}}$ states (C12 and F) and the WAEN value (CC) are frozen until $\overline{\text{AS}}$ is

recognized as deasserted. The $\overline{TA}$ signal driven by the UPM also remains in its programmed state until $\overline{AS}$ is negated. This allows $\overline{TA}$ to be used as an asynchronous handshake signal by programming UTA = 0 in the same RAM word in which WAEN = 1. If this is done, $\overline{TA}$ can be used to signal that $\overline{AS}$ should deassert (similar to $\overline{DTACK}$ in the 68000 bus).

The wait state is exited when $\overline{AS}$ is negated, at which point all external signals controlled by the UPM are driven high asynchronously from the $\overline{AS}$ deassertion. External signals are driven in this state until the LAST bit is set in a RAM word. The TODT bit is relevant only in words read by the UPM after $\overline{AS}$ is negated.

For a comprehensive discussion of external bus interfacing, see Section 15.8, “External Master Support.”

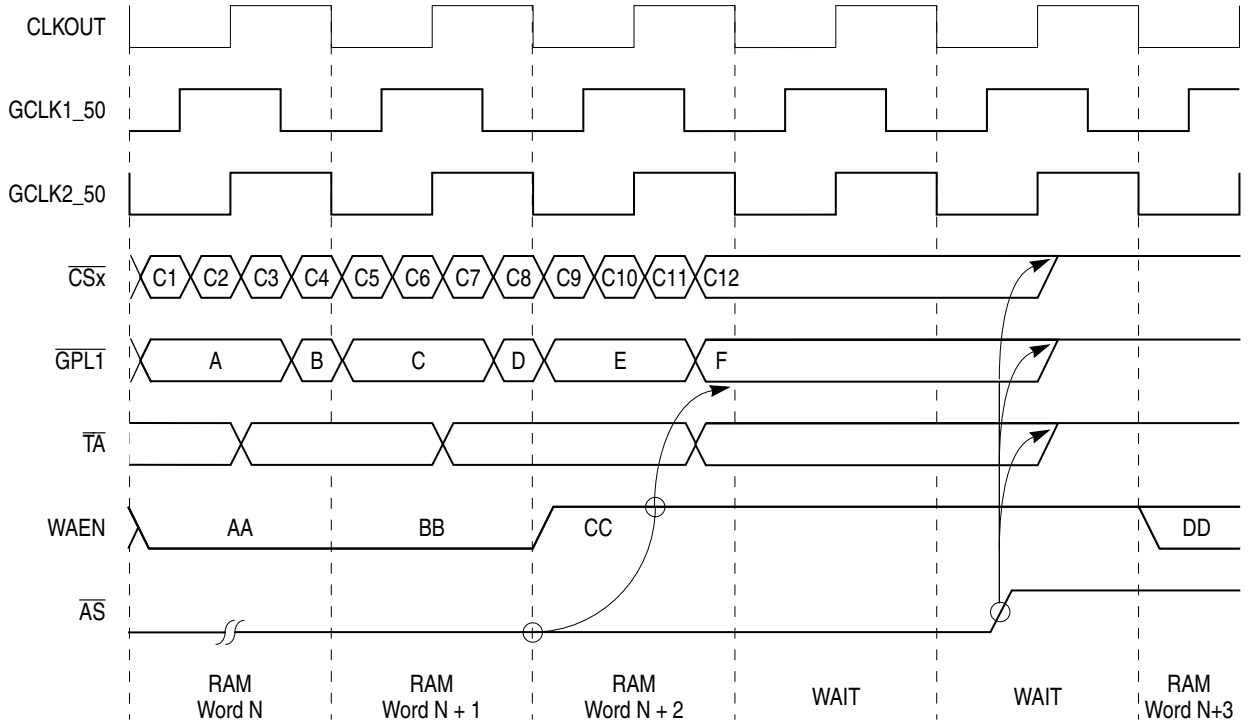


Figure 15-46. Wait Mechanism Timing for an External Asynchronous Master

15.7 Handling Devices with Slow or Variable Access Times

The memory controller provides two ways to interface with slave devices that are very slow (access time is greater than the maximum allowed by the user programming model) or cannot guarantee a predefined access time (for example some FIFO, hierarchical bus interface, or dual-port memory devices).

These mechanisms are as follows:

- The wait mechanism—Used only in accesses controlled by the UPM. MAMR[GPLA4DIS] and MBMR[GPLB4DIS] enable this mechanism.
- The external $\overline{\text{TA}}$ mechanism is used only in accesses controlled by the GPCM. OR*x*[SETA] specifies whether $\overline{\text{TA}}$ is generated internally or externally.

The following sections describe how the two mechanisms work.

15.7.1 Hierarchical Bus Interface Example

Assume that the CPU initiates a local-bus read cycle that addresses main memory connected to the system bus. The hierarchical bus interface accepts local bus requests and generates a read cycle on the system bus. The programmer cannot predict when valid data can be latched by the CPU because a DMA device may be occupying the system bus.

- The wait solution (UPM)—The external module asserts UPWAIT to the memory controller to indicate that data is not ready. The memory controller synchronized this signal because the wait signal is asynchronous. As a result of the wait signal being asserted, the UPM enters a freeze mode at the falling edge of CLKOUT upon encountering the WAEN bit being set in the UPM word. The UPM stays in that state until UPWAIT is negated. After UPWAIT is negated, the UPM continues executing from the next entry to the end of the pattern (LAST bit is set).
- The external $\overline{\text{TA}}$ solution (GPCM)—The bus interface module asserts $\overline{\text{TA}}$ to the memory controller when it can sample data.

15.7.2 Slow Devices Example

Assume the CPU initiates a read cycle from a device whose access time exceeds the maximum allowed by the user programming model.

- The wait solution (UPM)—The CPU generates a read access from the slow device. The device in turn asserts the wait signal as long as the data is not ready. The CPU samples data only after the wait signal is negated.
- The external $\overline{\text{TA}}$ solution (GPCM)—The CPU generates a read access from the slow device, which must generate the synchronous $\overline{\text{TA}}$ when it is ready.

15.8 External Master Support

The memory controller supports internal and external bus masters. Accesses from the core or the CPM are considered internal; accesses from an external bus master are external. External bus master support is available only if enabled in the SIU module configuration register (SIUMCR), described in Section 10.4.2. There are two types of external bus masters:

- Synchronous bus masters synchronize with CLKOUT and may or may not use the MPC855T memory controller to access a slave.

- Asynchronous bus masters use an address strobe signal ($\overline{AS}$) that handshakes with the MPC855T memory controller to access a slave device or bypass the memory controller to perform the slave access.

15.8.1 Synchronous External Masters

Synchronous masters initiate a transfer by asserting $\overline{TS}$. A[0–31], RD/$\overline{WR}$, $\overline{BURST}$, and TSIZ must be stable before the rising edge of CLKOUT after $\overline{TS}$ is asserted and until the last $\overline{TA}$ is negated. Because the external master operates synchronously with the MPC855T, meeting setup and hold times for all inputs associated with the rising edge of CLKOUT is critical. To support synchronous mode using the memory controller, SIUMCR[SEME] must be set. When $\overline{TS}$ is asserted, the memory controller compares the address with each of its valid banks. If a match is found, control signals to the slave are generated and $\overline{TA}$ is supplied to the external master. If SEME = 0, the memory controller is bypassed and the external synchronous master must provide control signals to the slave. See Figure 15-47.

15.8.2 Asynchronous External Masters

Asynchronous masters initiate transfers by driving the address bus and asserting $\overline{AS}$. A[0–31], RD/$\overline{WR}$, and TSIZ must have a proper setup time before $\overline{AS}$ is asserted. To support asynchronous mode, SIUMCR[AEME] must be set. The memory controller synchronizes $\overline{AS}$ assertion to its internal clock and generates control signals to the slave device. When $\overline{AS}$ is synchronized, the memory controller compares the address with each of its defined valid banks; if a match is found, control signals to the slave are generated and $\overline{TA}$ is supplied to the external master. All control signals to the memory device and $\overline{TA}$ are negated with the negation of $\overline{AS}$. If AEME = 0, the memory controller is bypassed and the external asynchronous master must provide control signals to the slave. In this mode, the MPC855T's $\overline{AS}$ signal cannot be used as an input. See Figure 15-48.

15.8.3 Special Case: Address Type Signals for External Masters

The AT signals are not sampled on the external bus for external master accesses. When external masters access slaves on the bus, the internal AT[0–2] signals reaching the memory controller are forced to '100'. The user should ensure this access matches the BR*x*[AT]. It is masked by OR*x*[ATM].

15.8.4 UPM Features Supporting External Masters

The following sections provide information on the UPM features that support external masters.

15.8.4.1 Address Incrementing for External Synchronous Bursting Masters

BADDR[28–30] should be used to generate addresses to memory devices for burst accesses. They duplicate the value of A[28–30] when an internal master initiates an external bus transaction. When an external master initiates an external bus transaction, they reflect the value of A[28–30] on the first clock cycle of the memory access; these signals are latched by the memory controller and on subsequent clock cycles, BADDR[28–30] increments as programmed in the UPM.

15.8.4.2 Handshake Mechanism for Asynchronous External Masters

A wait mechanism in the UPM supports handshaking for external asynchronous masters. This is provided with an $\overline{AS}$ input signal and the WAEN bit in the UPM RAM words. See Section 15.6.4.11, “The Wait Mechanism (WAEN).”

15.8.4.3 Special Signal for External Address Multiplexer Control

If external masters exist in the system with the MPC855T, address multiplexing (for DRAM for example) must be implemented in external logic. To control this external multiplexer, special features have been added to $\overline{GPL5}$. See Section 15.6.4.4, “General-Purpose Signals (GxTx, G0x).”

15.8.5 External Master Examples

The following sections provide external master examples.

15.8.5.1 External Masters and the GPCM

The following figures show examples of external masters’ interaction with the GPCM. Note that synchronous and external masters behave differently. Synchronous external masters behave like internal masters, except for an extra clock cycle at the beginning of the access required for address decode. Asynchronous external masters behave as described in Section 15.5.3, “External Asynchronous Master Support.”

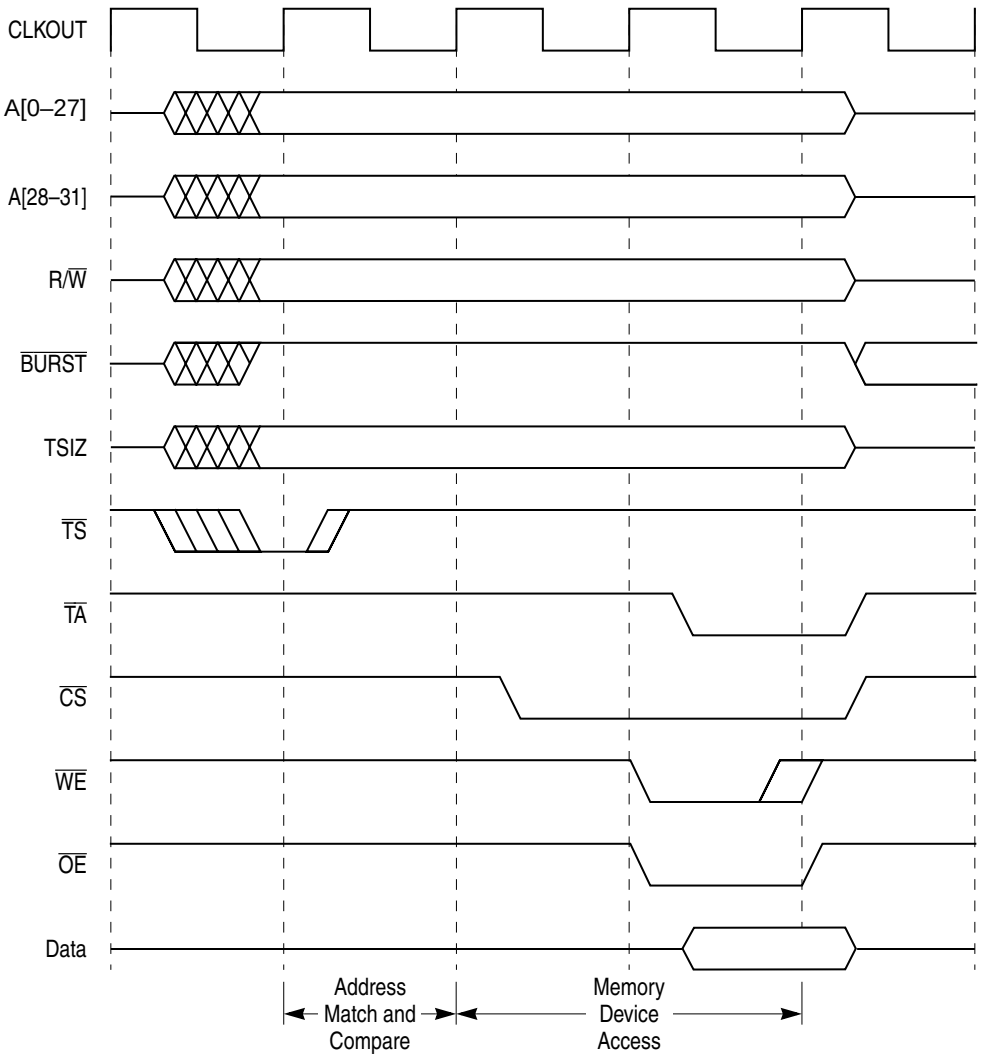


Figure 15-47. Synchronous External Master Access

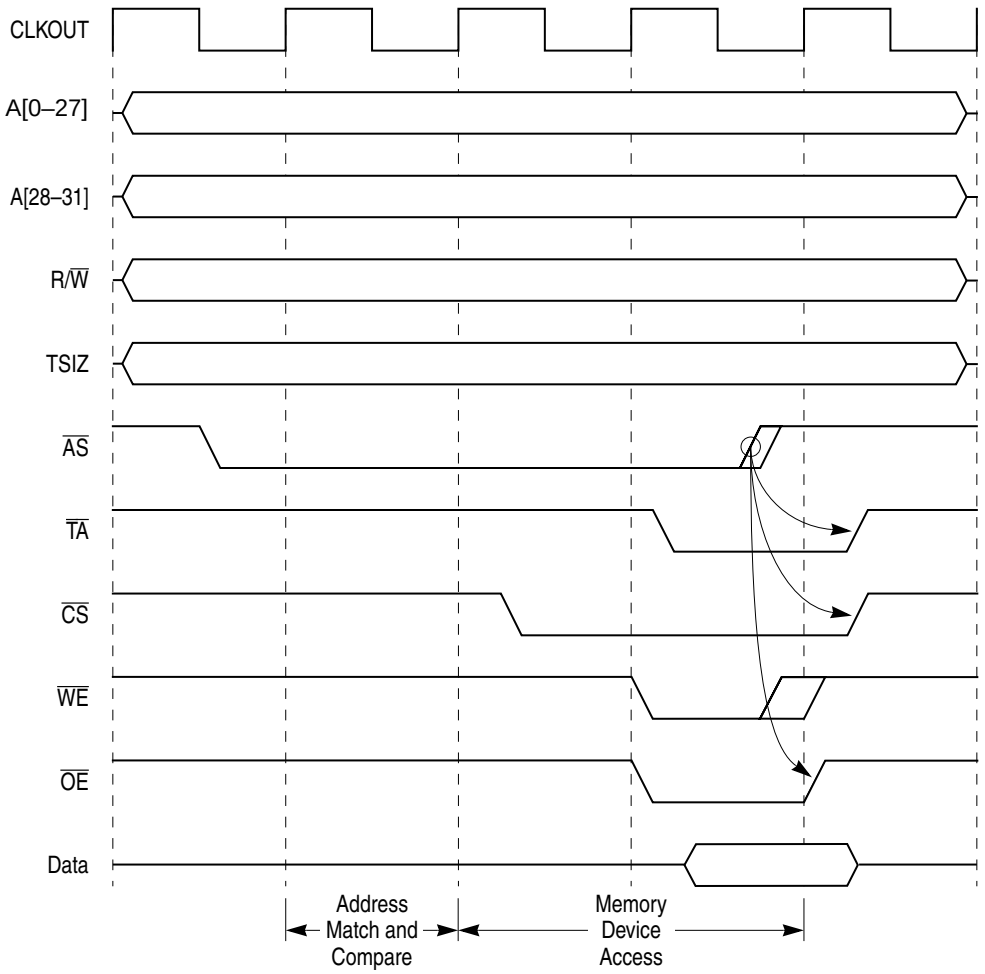


Figure 15-48. Asynchronous External Master Access

15.8.5.2 External Masters and the UPM

Figure 15-49 shows a synchronous interconnection in which an external master and the MPC855T can share access to a DRAM bank. Notice that $\overline{CS1}$, UPMA, and $\overline{GPL\_A5}$ were chosen to help control DRAM bank accesses. To perform burst accesses initiated by the external master or MPC855T using this configuration, BADDR[28–30] connects to the multiplexer controlled by $\overline{GPL\_A5}$. Figure 15-50 shows the timing behavior of$\overline{GPL_A5}$, BADDR, and other control signals when an external master initiates a burst read access. The state of $\overline{GPL\_A5}$ in the first clock cycle of the memory device access is determined by the value of the corresponding OR*x*[G5LS]. In this example, the accessed critical word is addressed at BADDR[28–29] = 10, which then increments and wraps around to the word before the critical word (01) for subsequent beats of this burst access.

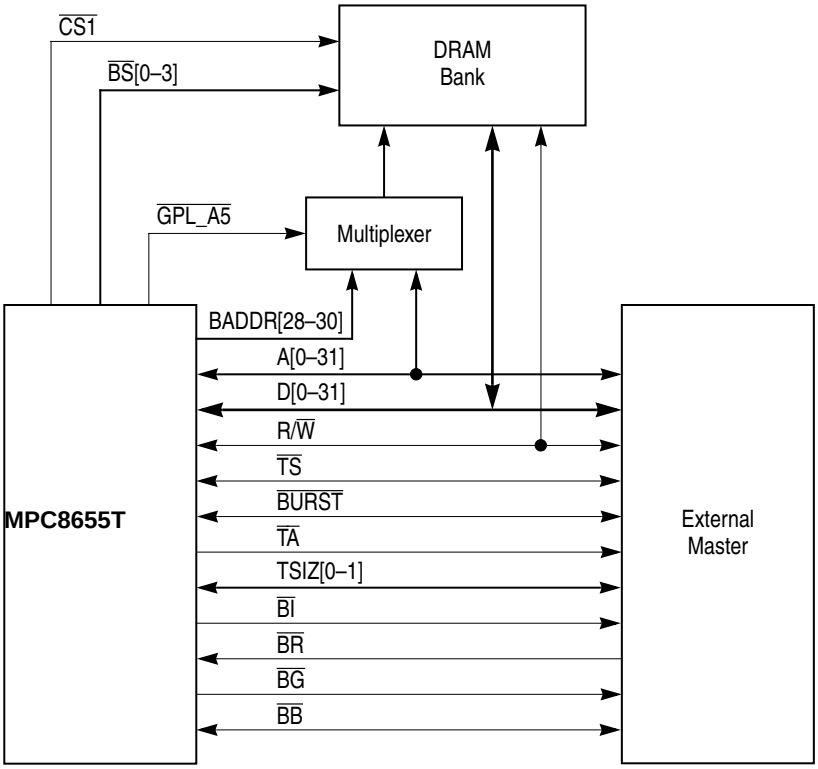
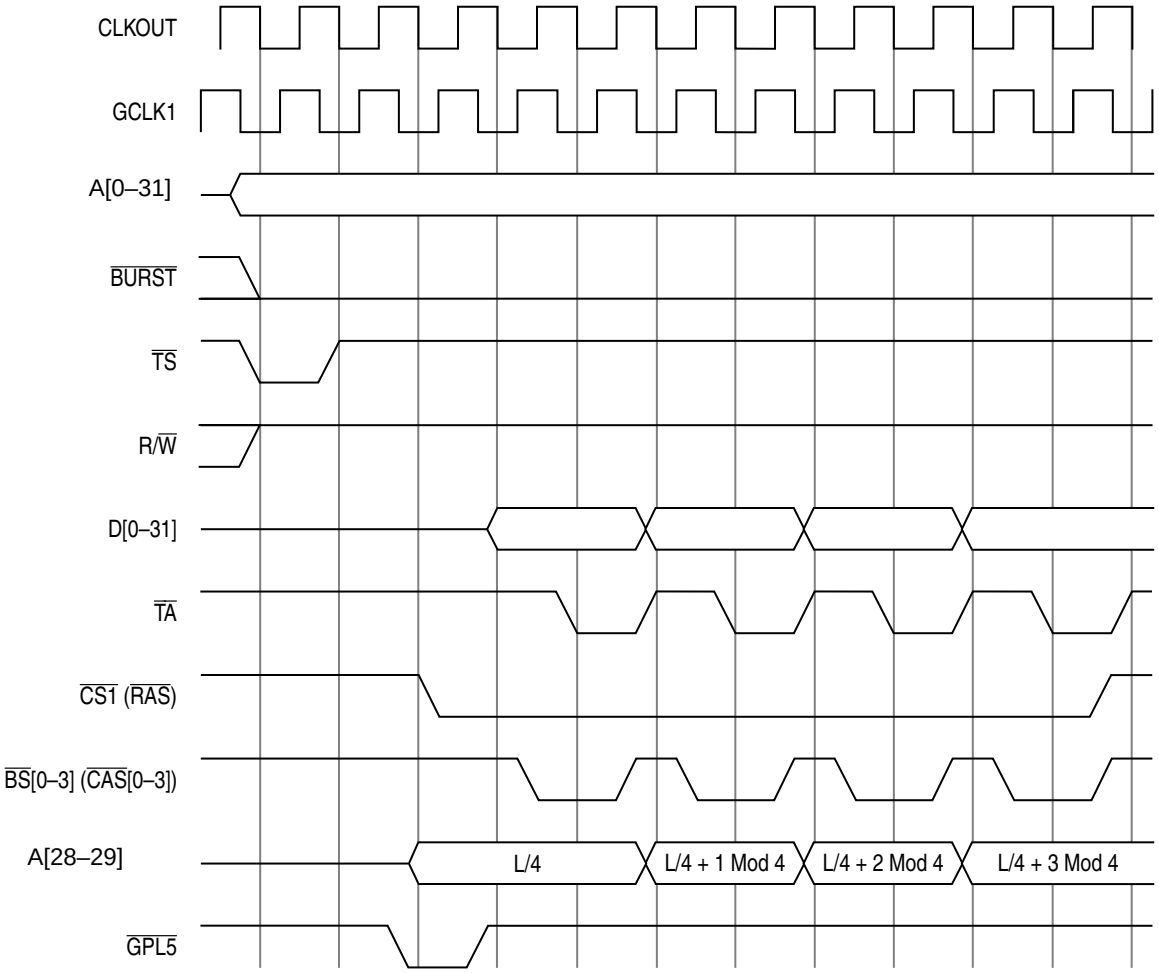


Figure 15-49. Synchronous External Master Interconnect Example



cst4Bit 0000000000
cst1Bit 1000000000
cst2Bit 2000000001
cst3Bit 3000000001
bst4Bit 4110101010
bst1Bit 5100000000
bst2Bit 6101010101
bst3Bit 7101010101
g0l0Bit 8
• •
• •
• •
g5t4Bit 20011111111
g5t3Bit 21011111111
–Bit 22
–Bit 23
loopBit 24000000000
exenBit 25001010100
amx0Bit 2600000000X
amx1Bit 2700000000X
naBit 2800101010X
utaBit 29101010101

Figure 15-50. Synchronous External Master: Burst Read Access to Page Mode DRAM

Figure 15-51 shows an asynchronous interconnection in which an external master and the MPC855T can share access to a DRAM bank. Notice that $\overline{CS1}$, UPMA, and $\overline{GPL\_A5}$ were chosen to control DRAM bank accesses. Figure 15-52 shows the timing behavior of $\overline{GPL\_A5}$ and other control signals when an external master to a DRAM bank initiates a single-beat read. The state of $\overline{GPL\_A5}$ in the first clock cycle of the memory device access is determined by the value of the corresponding OR*x*[G5LS].

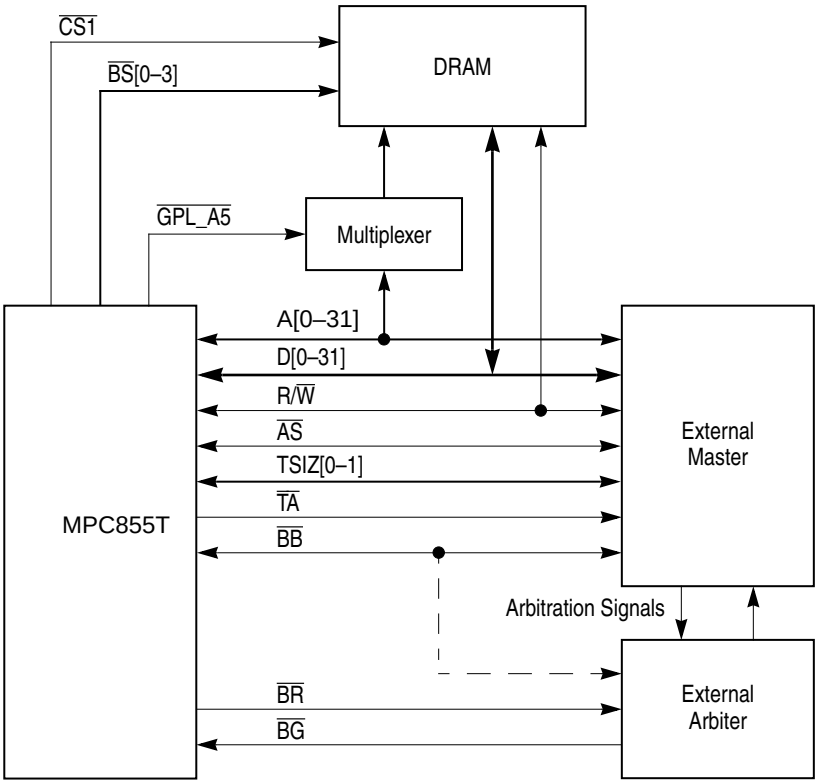
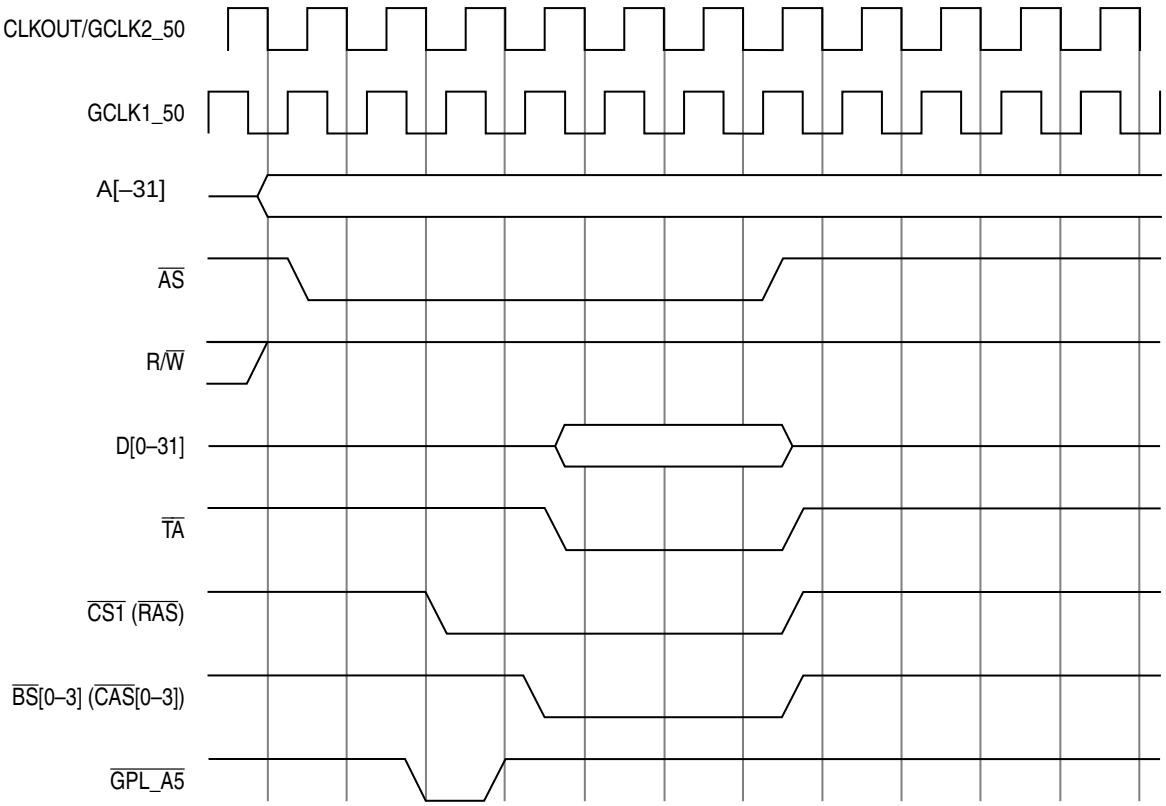


Figure 15-51. Asynchronous External Master Interconnect Example



cst4Bit 000000 cst1Bit 100000 cst2Bit 200001 cst3Bit 300001											
bst4Bit 411000 bst1Bit 510000 bst2Bit 610001											
bst3Bit 710001 • • • • • •											
g4t4Bit 18 g4t3/WAENBit 1901110											
g5t4Bit 2001110 g5t3Bit 2101111											
–Bit 22 –Bit 23											
loopBit 2400000 exenBit 2500000 amx0Bit 2600000 amx1Bit 2700000 naBit 2800000											
utaBit 2910001											

Figure 15-52. Asynchronous External Master Timing Example

15.9 Memory System Interface Examples

The following examples show 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 any UPM. UPMA is used in the page mode example and UPMB is used in the extended data out example.

15.9.1 Page-Mode DRAM Interface Example

Figure 15-53 shows configuration for a 1-Mbyte, 32-bit wide memory system using four 256 Kbyte x 8-bit DRAMs. Also shown is the physical connection between UPMA and the page mode DRAM. $\overline{CS1}$ is connected to all $\overline{RAS}$ and is controlled by the base register. $\overline{BS\_A[0–3]}$ 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 25-MHz baud rate generator clock and the DRAM that requires a 512-cycle refresh every 8 ms.

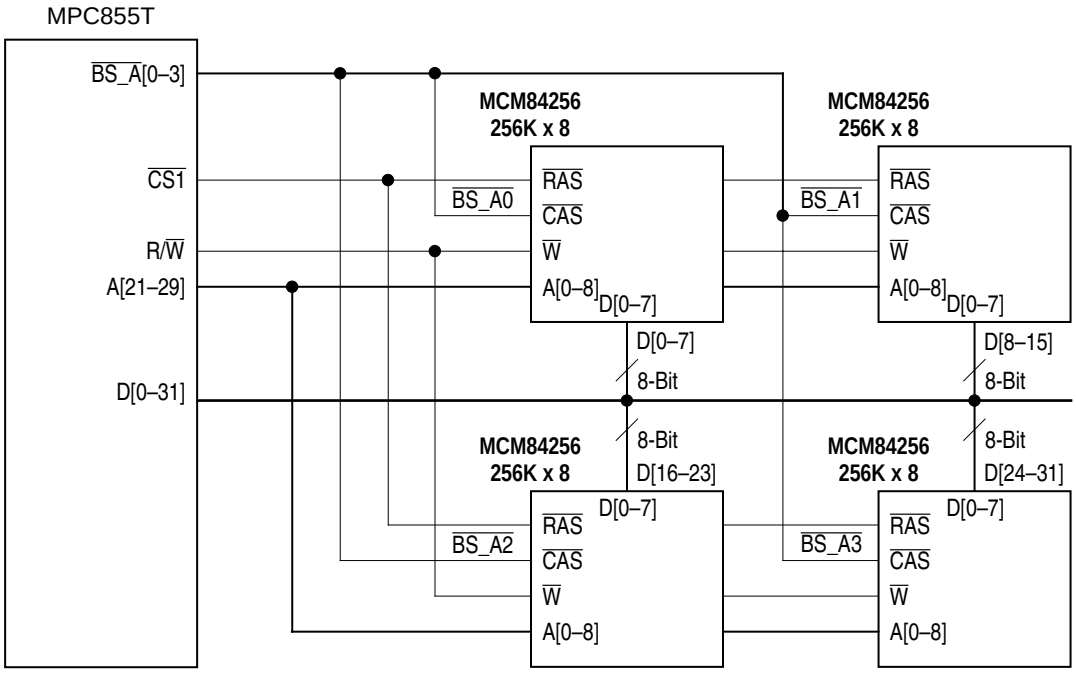


Figure 15-53. Page-Mode DRAM Interface Connection

Follow these steps to configure a system for page mode DRAM:

1. Determine the system architecture, which includes the MPC855T and the memory system as shown in the example in Figure 15-53.
2. Use the blank work sheet in Figure 15-70 to draw the timing diagrams for all the memory cycles. The timing diagrams in Figure 15-54 through Figure 15-62 can be used as a reference. Alternately, use the UPM860 or MCU unit applications for this. These applications are available at http:/www.motorola.com.

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 UPM parameters that control the memory system in the following sequence. For additional details, see Table 15-20.
   - Program the RAM array using MCR and MDR. The RAM word must be written into the MDR before a WRITE command is issued 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 chosen.
   - Use OR*x*[MS] to select the machine to control the cycles. Notice that OR*x*[SAM] 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 address multiplexing for the next clock cycle rather than the current one.
   - Program MAMR to select the number of columns and refresh timer parameters.

Table 15-20. UPMA Register Settings

Register	Field	Value	Comments
BR1	MS	10	Selects UPMA
	PS	00	Selects 32-bit bus width
	WP	0	Allows read and write accesses
MPTPR	PTP	0010_0000	Prescaler divided by two
MAMR	PTA	0000_1100	15.6 µs at a 25-MHz clock
	PTAE	1	Enables periodic timer A
	AMA	001	Selects nine column address pins
	DSA	01	Selects two disable timer clock cycles
	GPLA4DIS	0	Disables the UPWAITA signal
	RLFA	0011	Selects three loop iterations for read
	WLFA	0011	Selects three loop iterations for write
OR1	SAM	1	Selects column address on first cycle
	BIH	0	Supports burst accesses

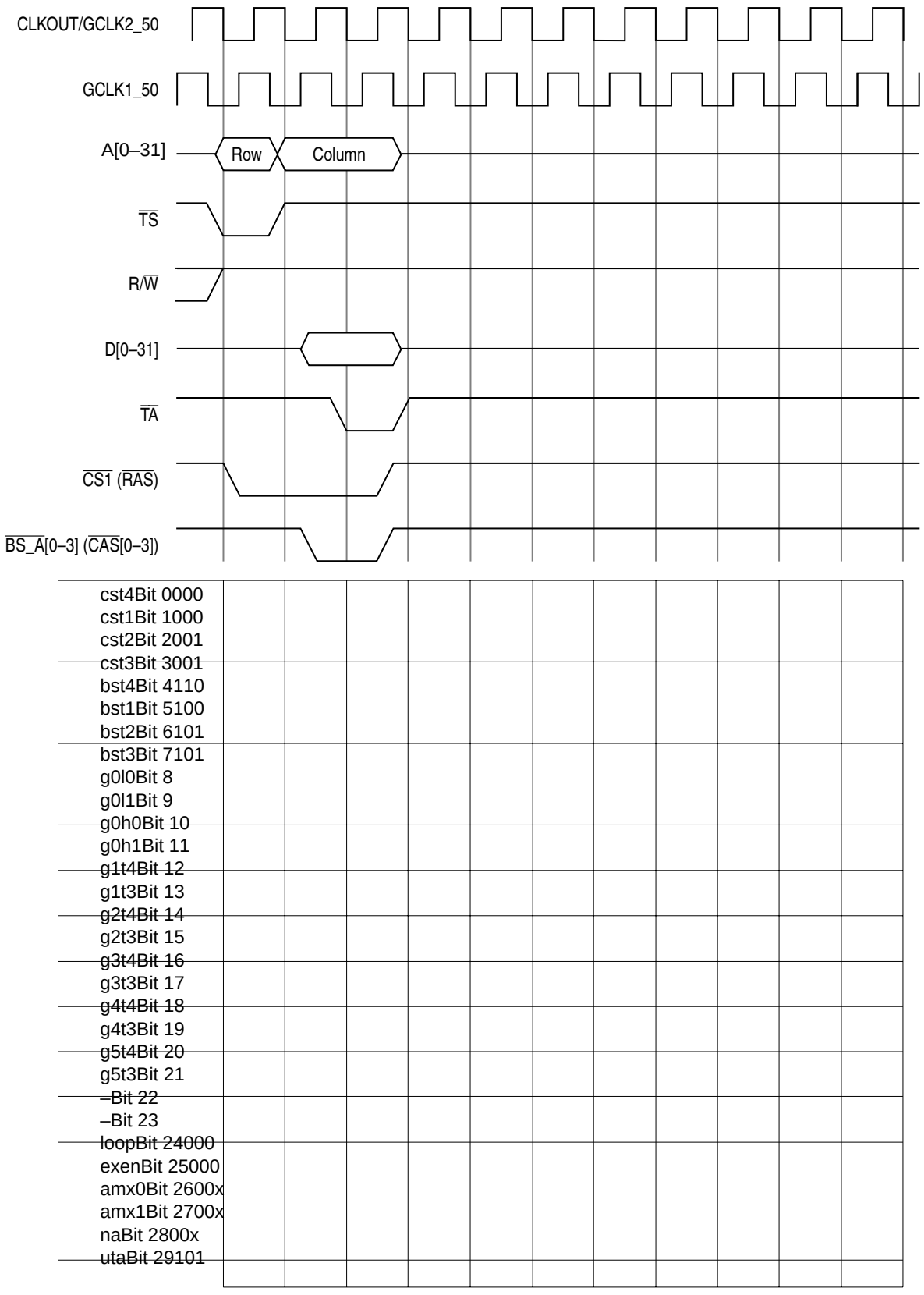


Figure 15-54. Single-Beat Read Access to Page-Mode DRAM

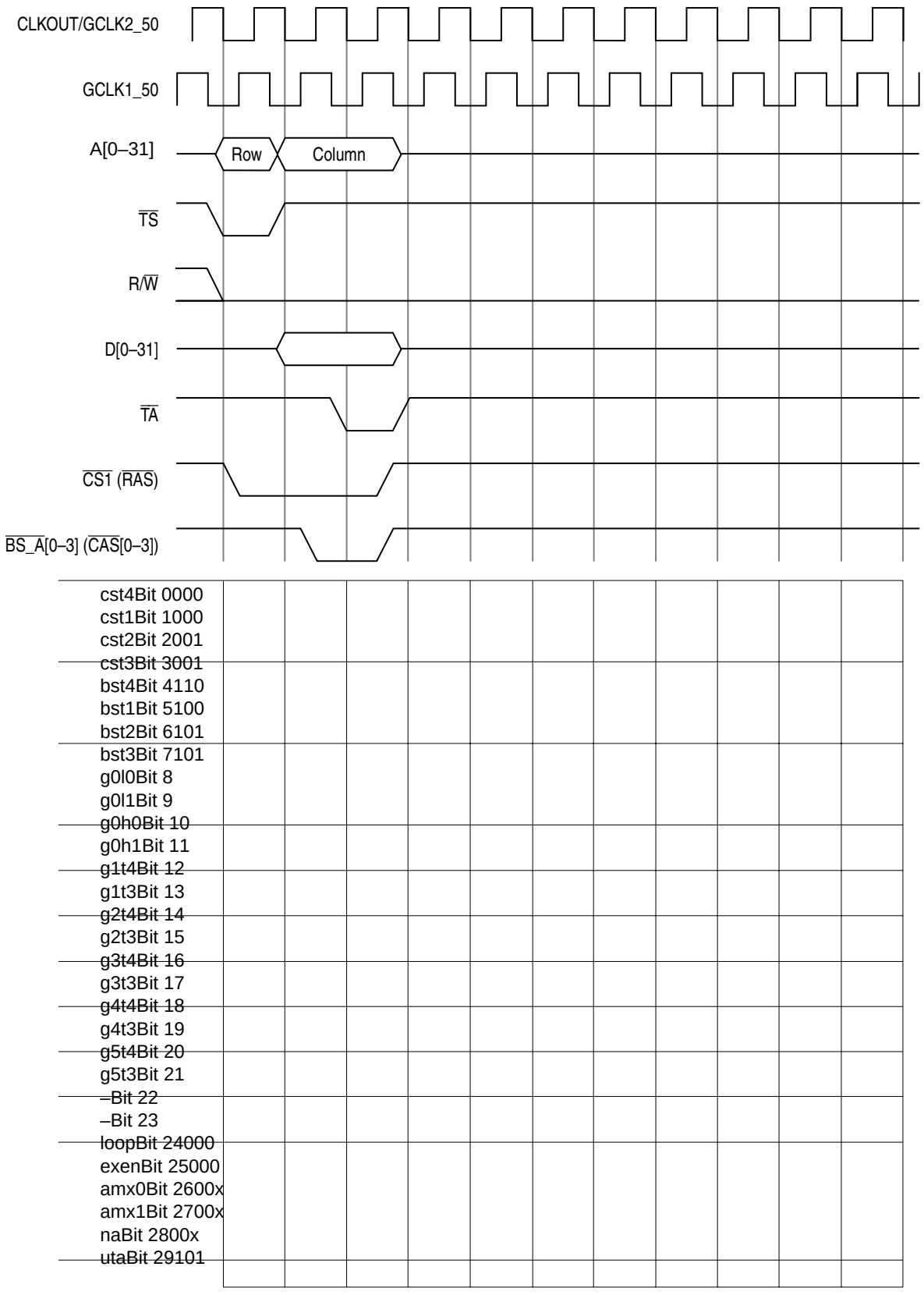


Figure 15-55. Single-Beat Write Access to Page Mode DRAM

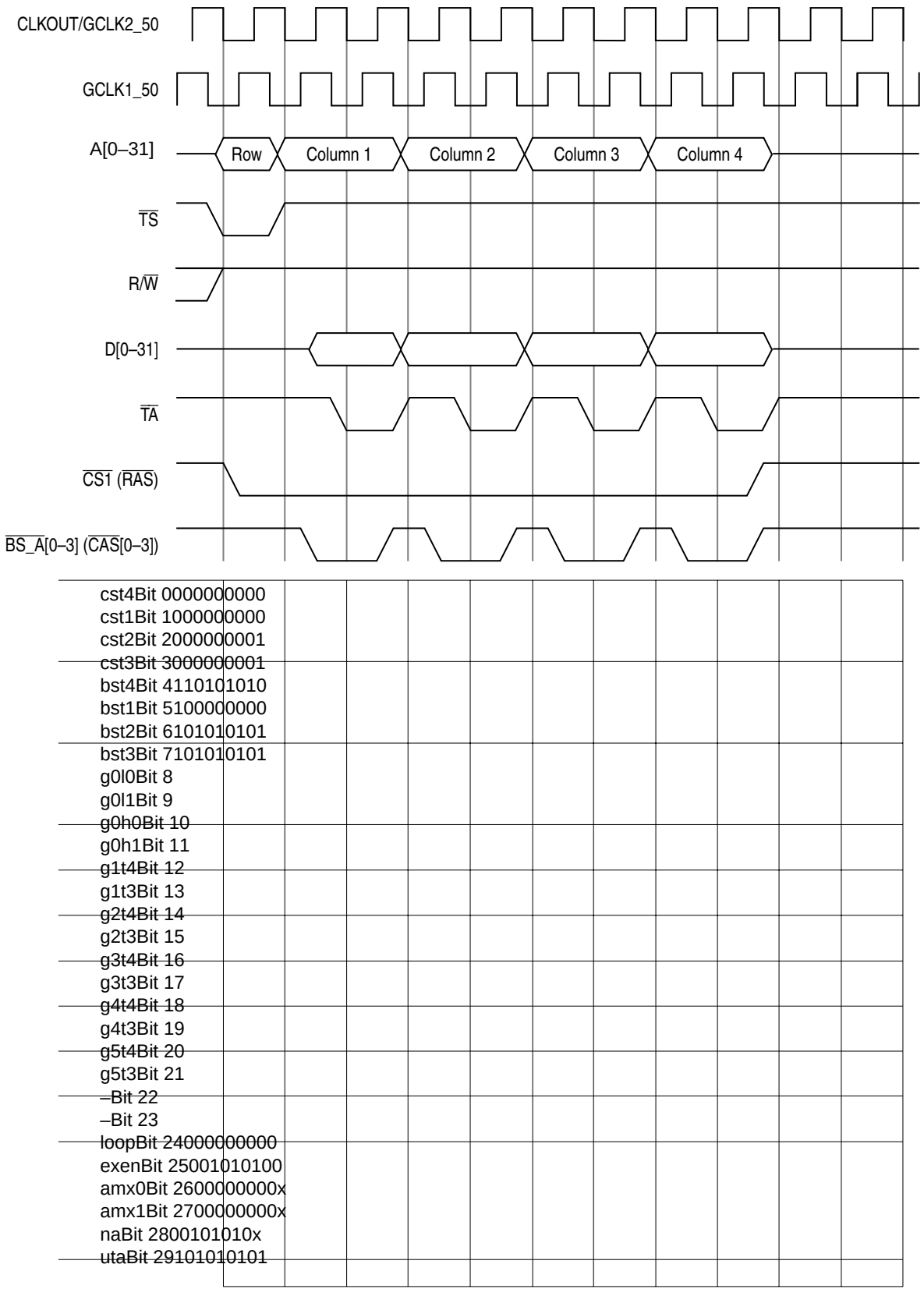


Figure 15-56. Burst Read Access to Page-Mode DRAM (No LOOP)

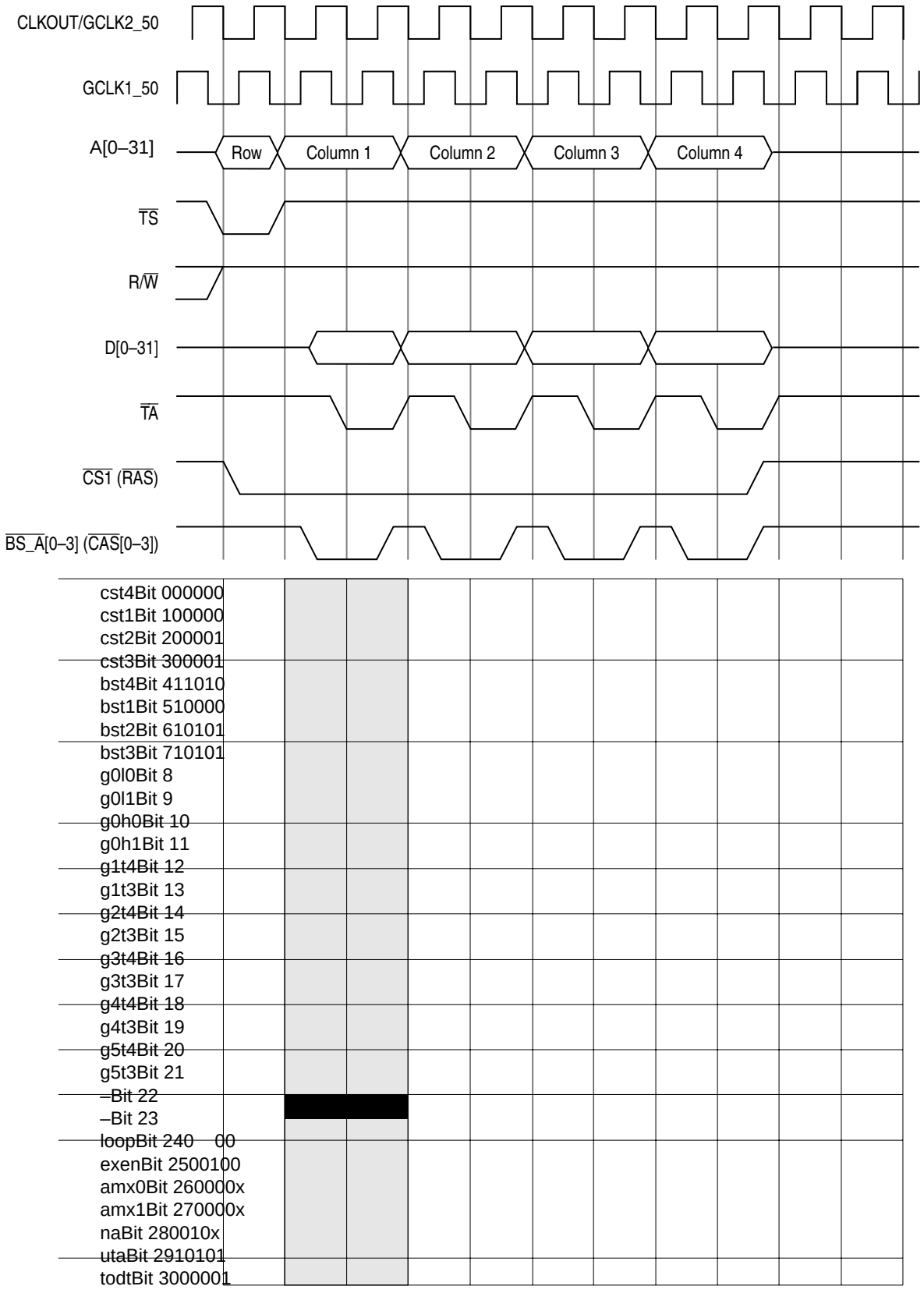


Figure 15-57. Burst Read Access to Page-Mode DRAM (LOOP)

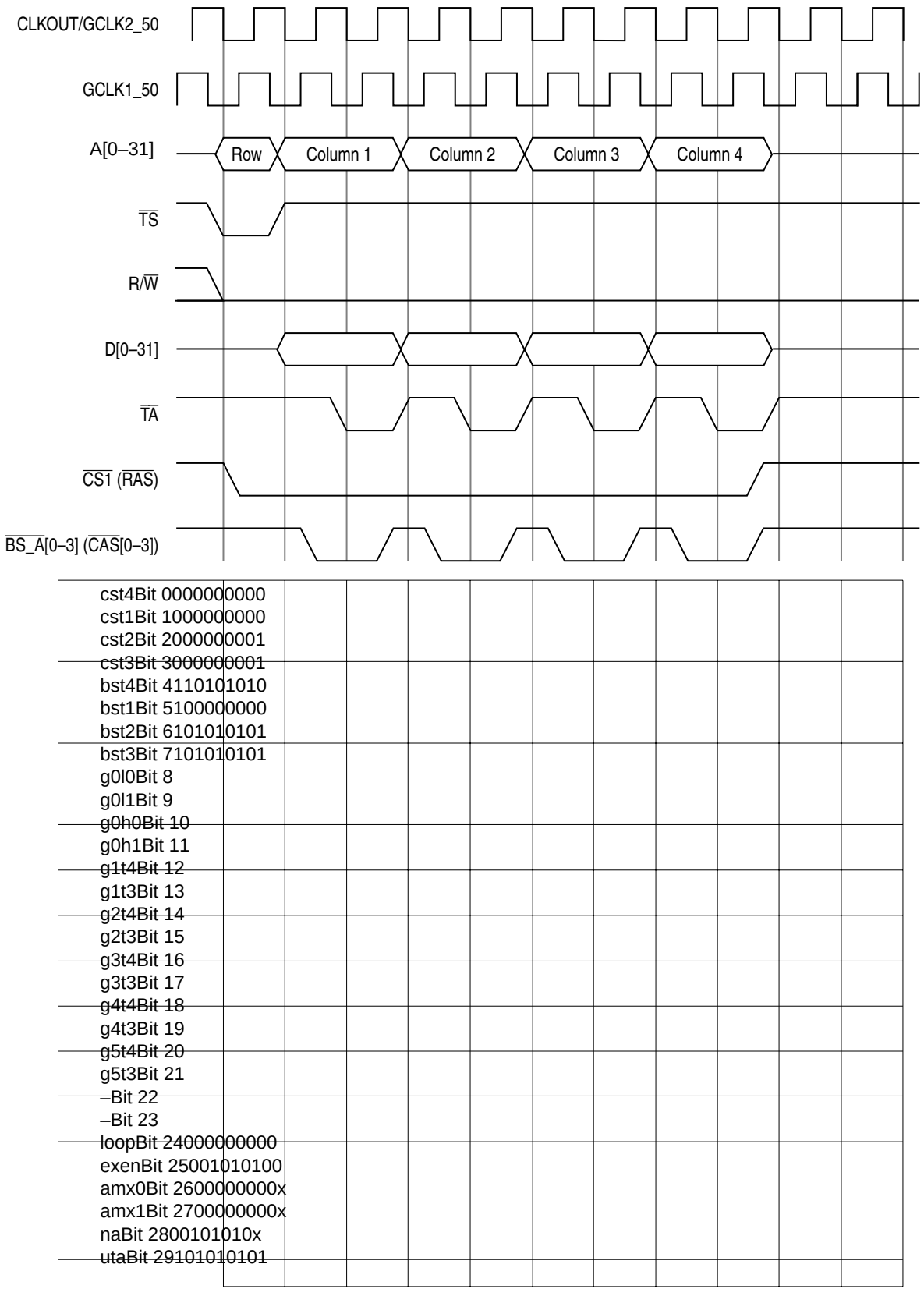


Figure 15-58. Burst Write Access to Page-Mode DRAM (No LOOP)

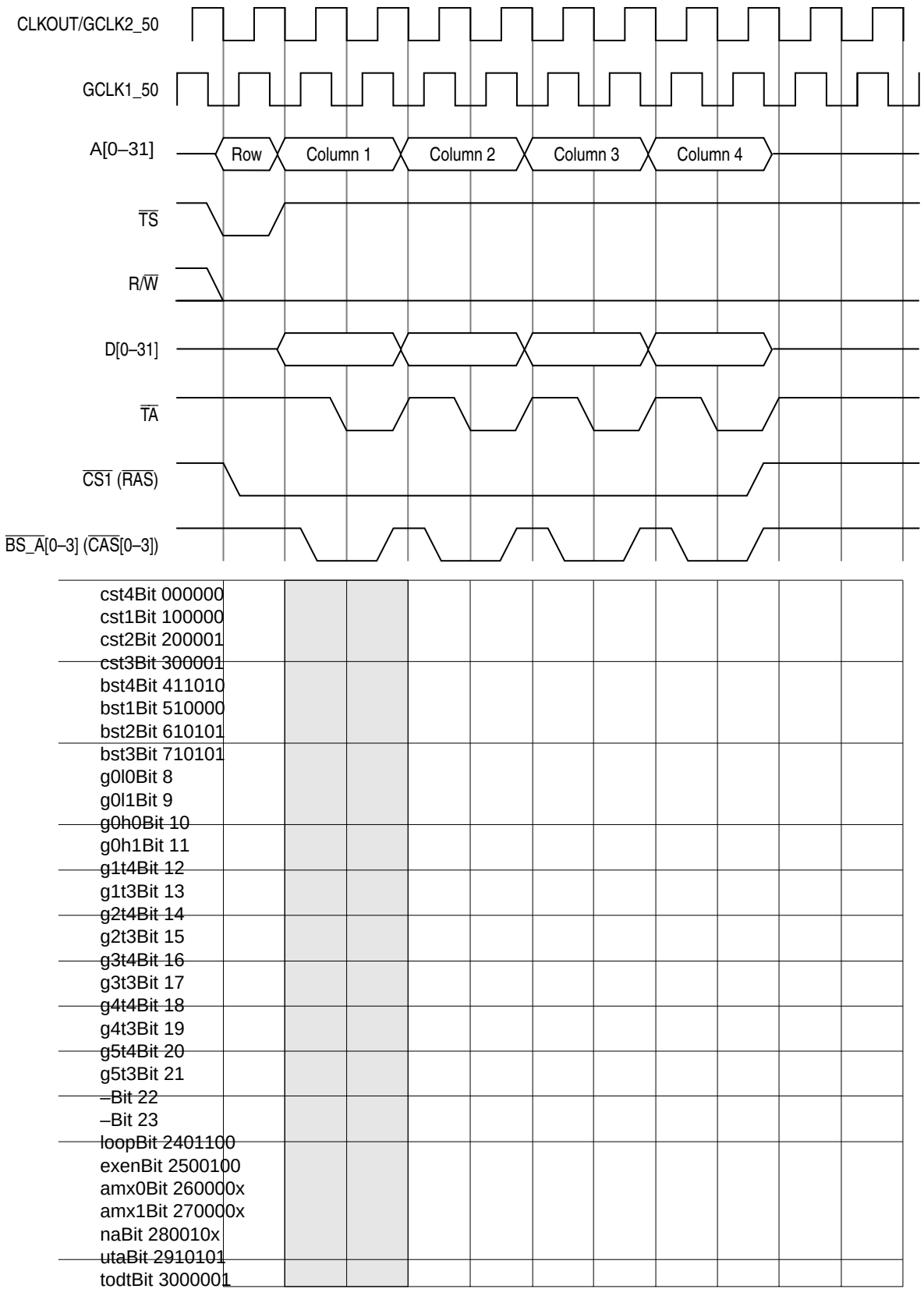


Figure 15-59. Burst Write Access to Page-Mode DRAM (LOOP)

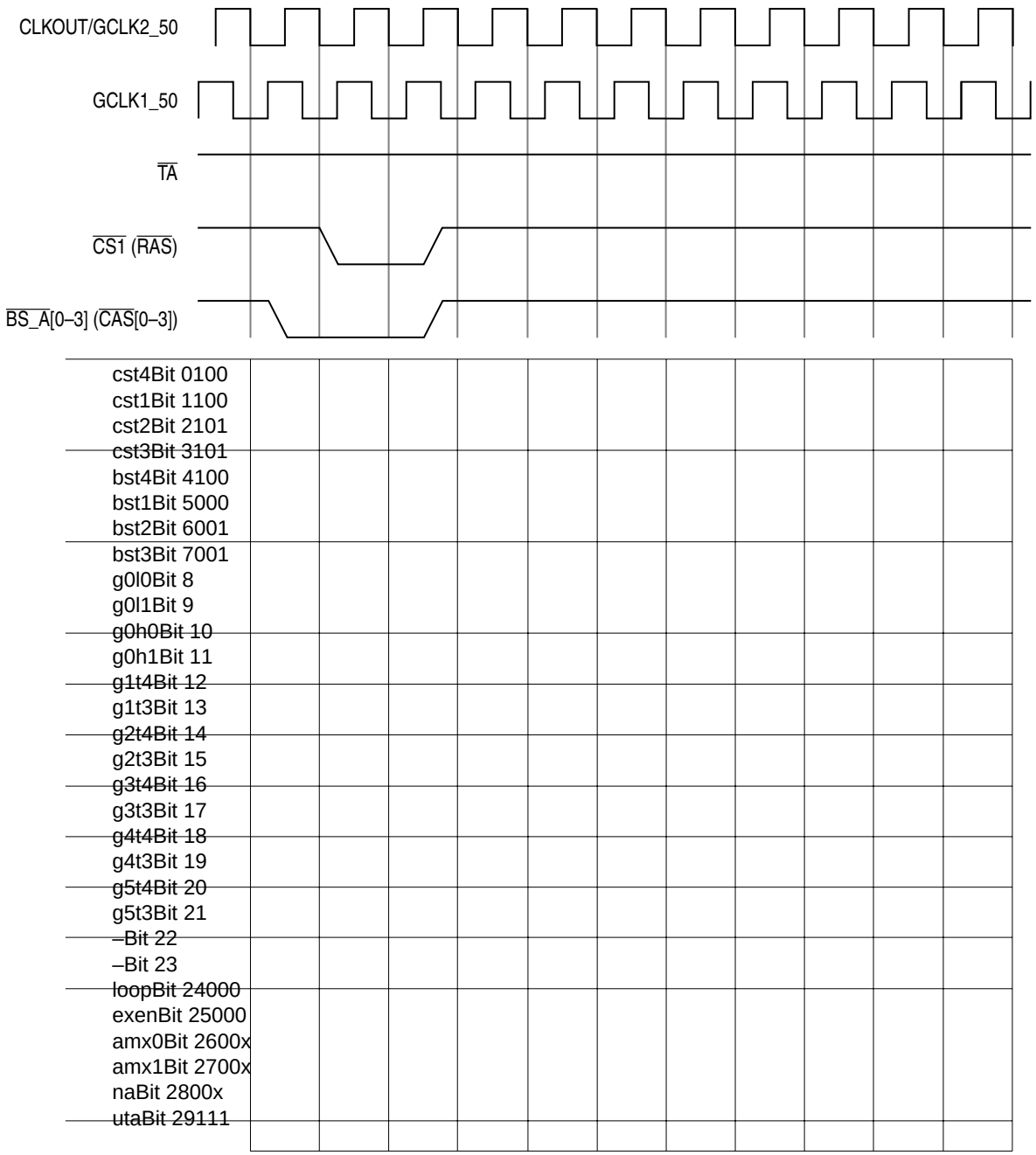


Figure 15-60. Refresh Cycle (CAS before RAS) to Page-Mode DRAM

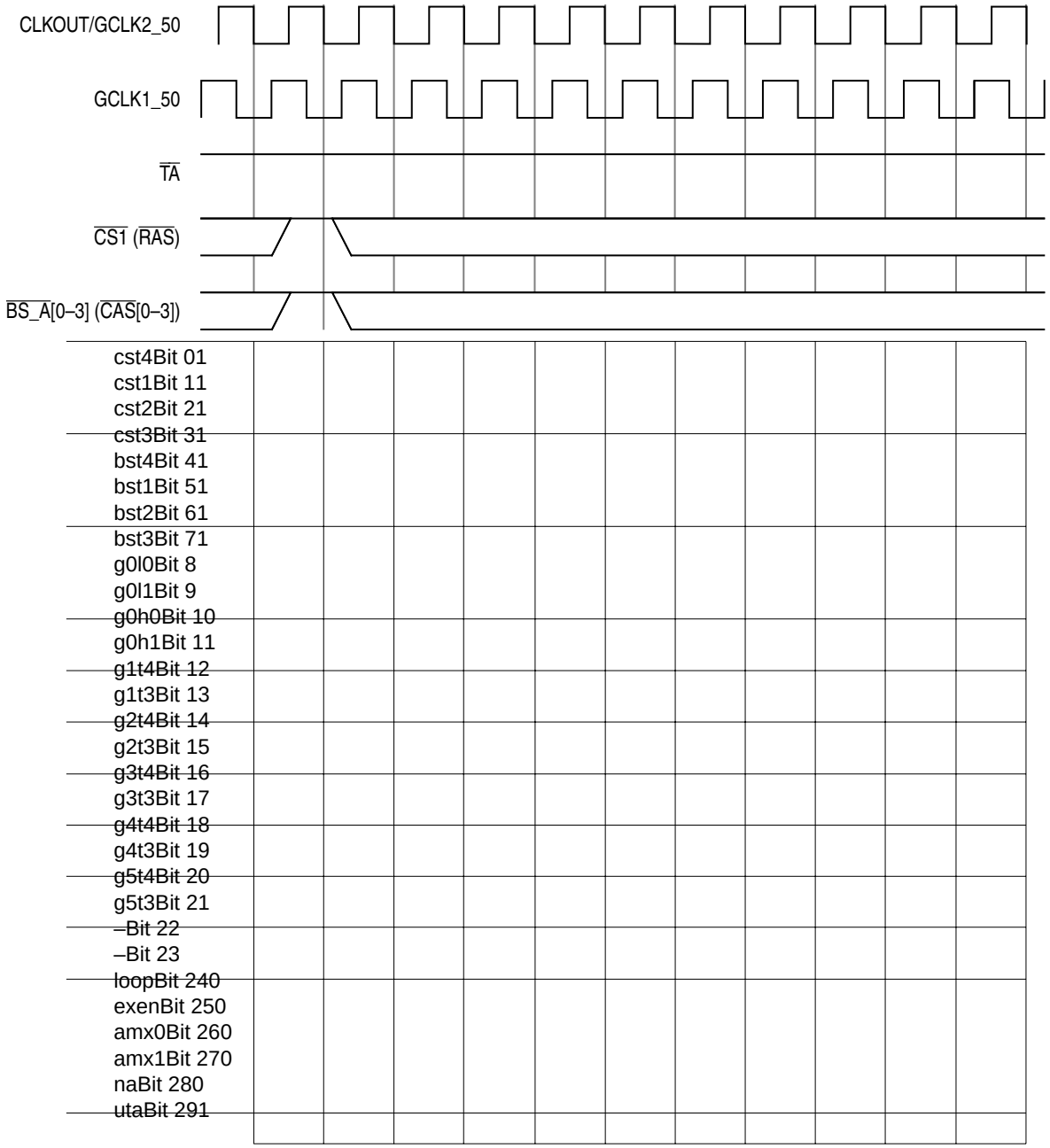


Figure 15-61. Exception Cycle

Page read accesses can be improved significantly by setting MAMR[GPLA4DIS] and ignoring GPL_A4. The processor samples the data bus at the falling edge of GCLK2_50 when TA is asserted. Figure 15-62 shows how to use this feature to change the burst read access to page mode DRAM (no loop). During the four consecutive data beats, TA is asserted to ensure a data transfer on every data clock. The figure also shows how the burst read access shown in Figure 15-56 of can be reduced from 9 to 6 cycles (for 32-bit port size). Cycles can be reduced by using faster DRAM or a slower system clock that meets the

DRAM access time. For a 16-bit port size memory, 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.

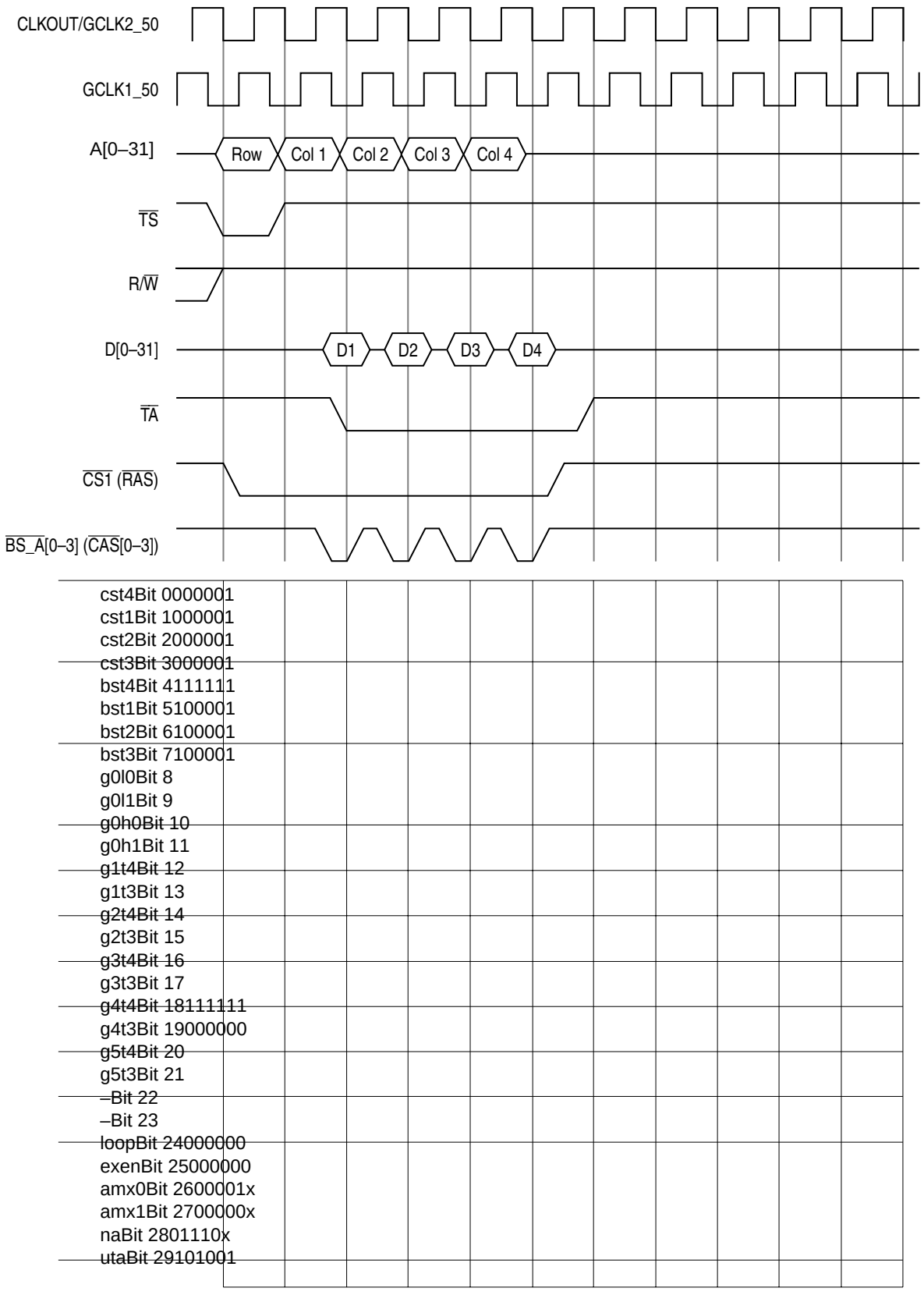


Figure 15-62. Optimized DRAM Burst Read Access

15.9.2 Page Mode Extended Data-Out Interface Example

Figure 15-63 shows the configuration for a 1-Mbyte, 32-bit wide memory system using two 256K x 16-bit page mode EDO DRAMs. Also shown is the physical connection between UPMB and the EDO DRAMs. The $\overline{CS2}$ signal controlled by BR*x* is connected to both $\overline{RAS}$ signals. The $\overline{BS\_B[0–1]}$ signals map to D[0–15] and $\overline{BS\_B[2–3]}$ map to D[16–31]. For this connection, $\overline{GPL\_B1}$ is connected to the memory device $\overline{OE}$ pins. The refresh rate calculation is based on a 25-MHz baud rate generator clock and the DRAM that requires a 512-cycle refresh every 8 ms.

This system has no external masters, and thus the MPC855T is configured to perform address multiplexing internally.

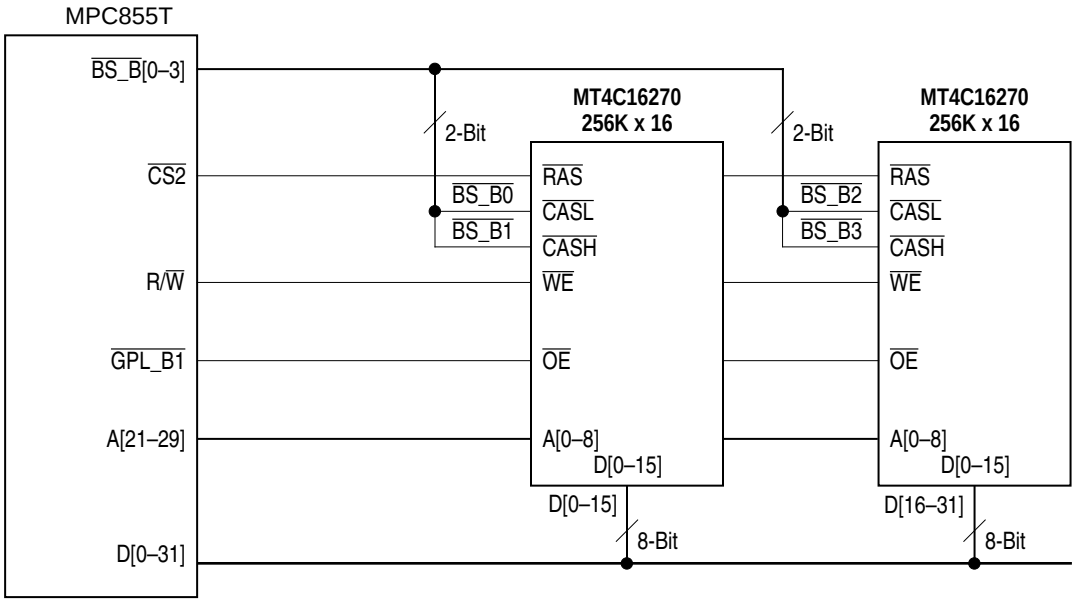


Figure 15-63. EDO DRAM Interface Connection

Follow these steps to configure a system for EDO DRAM:

1. Determine the system architecture, which includes the MPC855T and the memory system as shown in the example in Figure 15-64.
2. Use the blank work sheet in Figure 15-70 for timing diagrams. The timing diagrams in Figure 15-64 through Figure 15-69 can be used as a reference.
3. Translate the timing diagrams into RAM words for each memory access type. The bottom half of the figures show the RAM array contents that handle each of the possible cycles; each column represents a different word in the RAM array. A blank cell indicates a don’t care bit (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 15-20.

— Program the RAM array using MCR and MDR. The RAM word must be written into the MDR before a WRITE command is issued to the MCR. Repeat this step for all RAM word entries.

— Initialize OR*x* and BR*x* for the required DRAM device address mapping.

— OR*x*[MS] selects the machine to control clock cycles. Note that OR*x*[SAM] controls address multiplexing for the first cycle; subsequent cycles are controlled by UPM RAM words. Also note that the AMX field in the UPM RAM word controls address multiplexing for the next clock cycle and not the current one.

— Program MBMR to select the number of columns and refresh timer parameters.

Table 15-21. UPMB Register Settings

Register	Field	Value	Comments
BR2	MS	10	Selects UPMB
	PS	00	Selects 32-bit bus width
	WP	0	Allows read and write accesses
MPTPR	PTP	0000_0010	Prescaler divided by 32
MBMR	PTB	0000_1100	15.6 µs at a 25-MHz clock
	PTBE	1	Enables periodic timer B
	AMB	001	Selects nine column address pins
	DSB	01	Selects two disable timer clock cycles
	GPLB4DIS	0	Disables the UPWAITB signal
	RLFB	0011	Selects three loop iterations for read
	WLFB	0011	Selects three loop iterations for write
OR2	SAM	1	Selects column address on first cycle
	BIH	0	Supports burst accesses

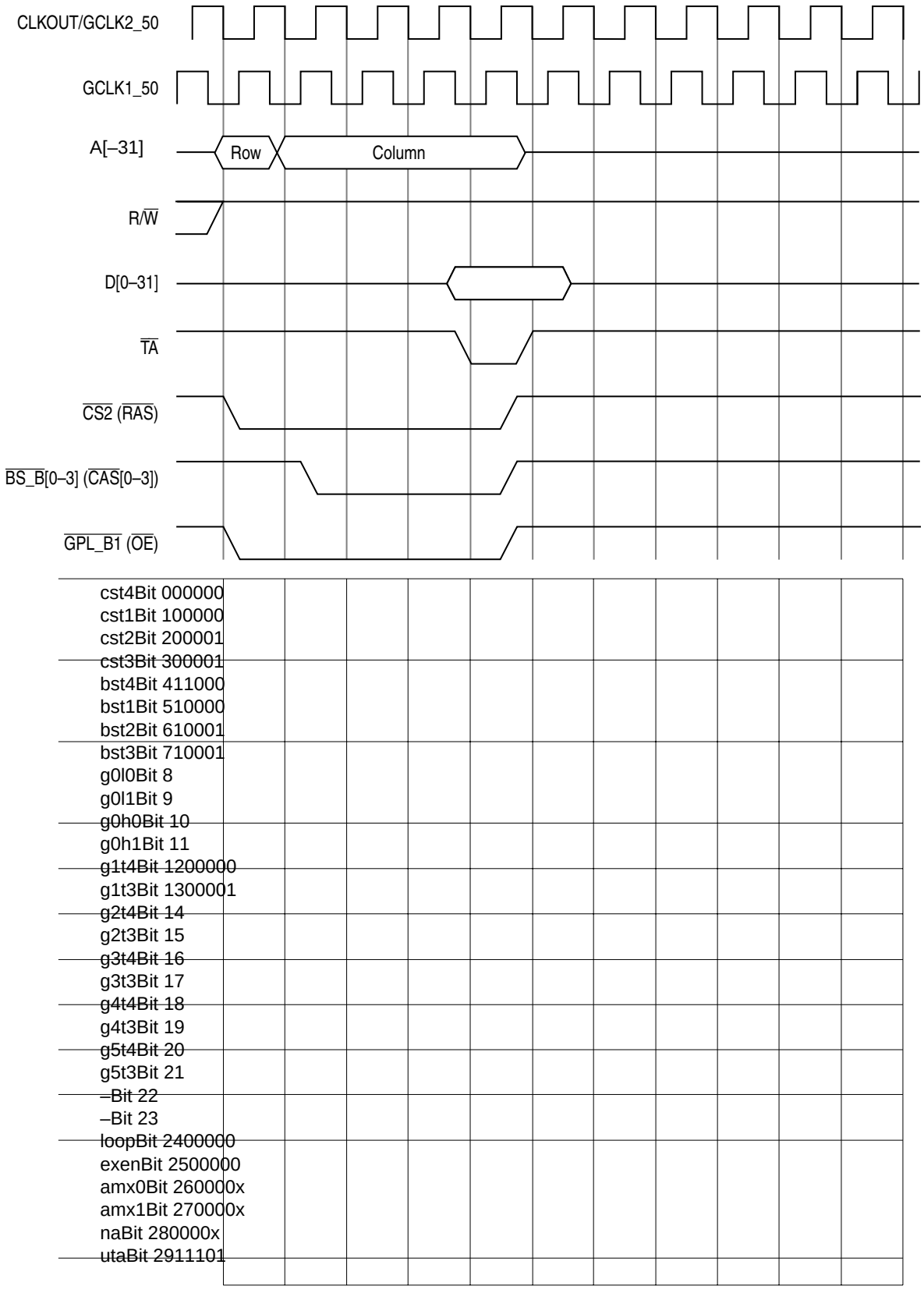


Figure 15-64. EDO DRAM Single-Beat Read Access

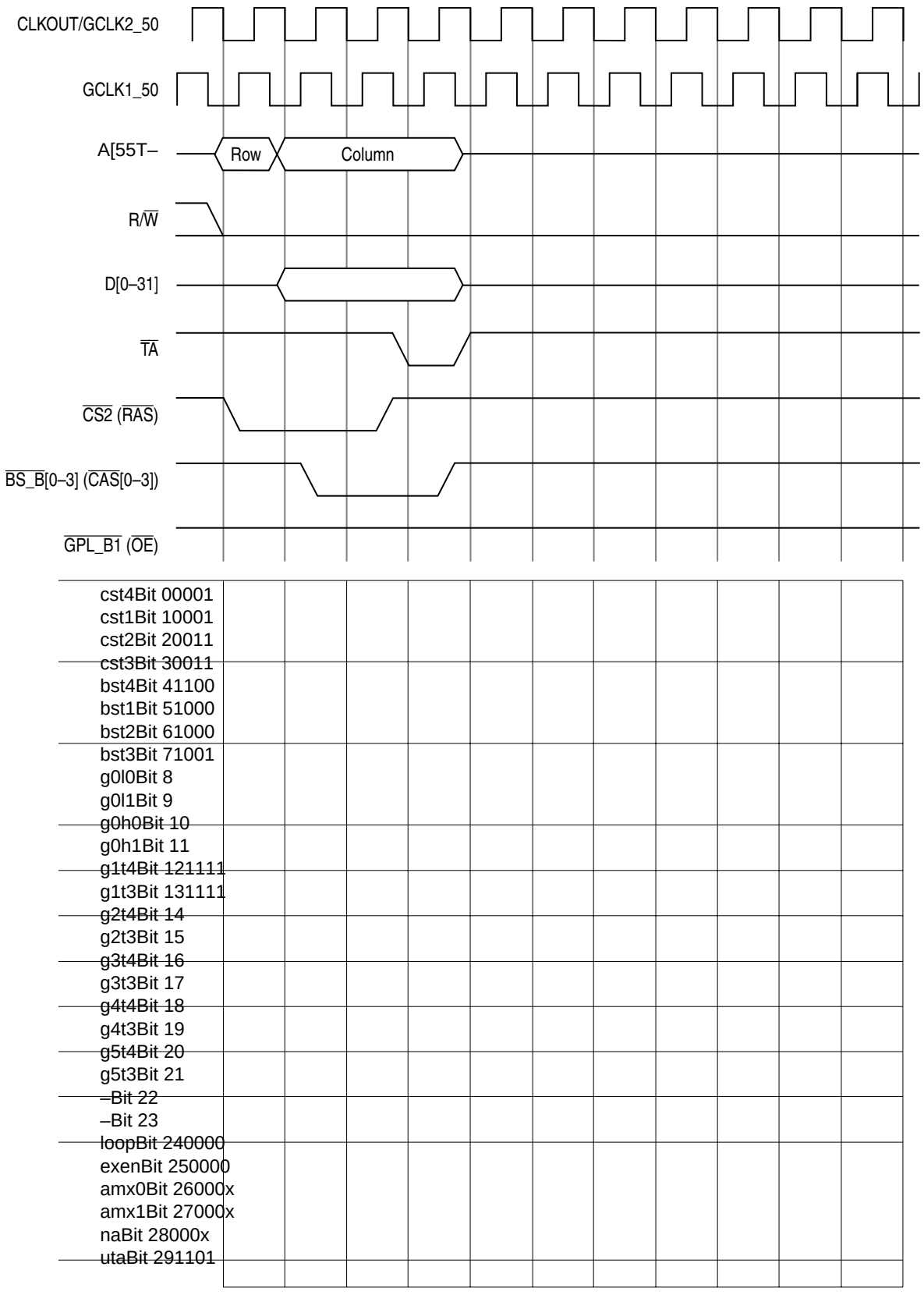


Figure 15-65. EDO DRAM Single-Beat Write Access

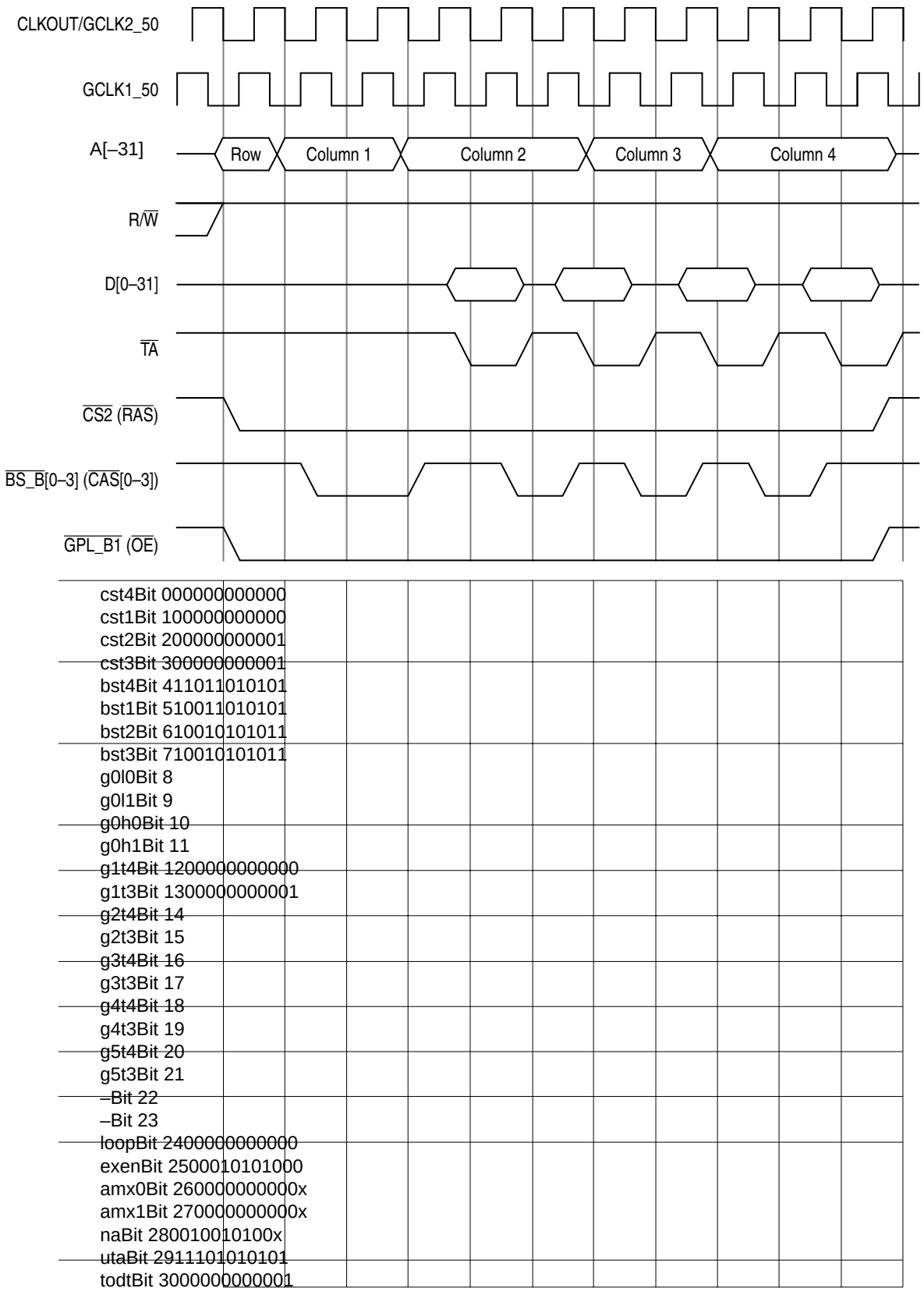


Figure 15-66. EDO DRAM Burst Read Access

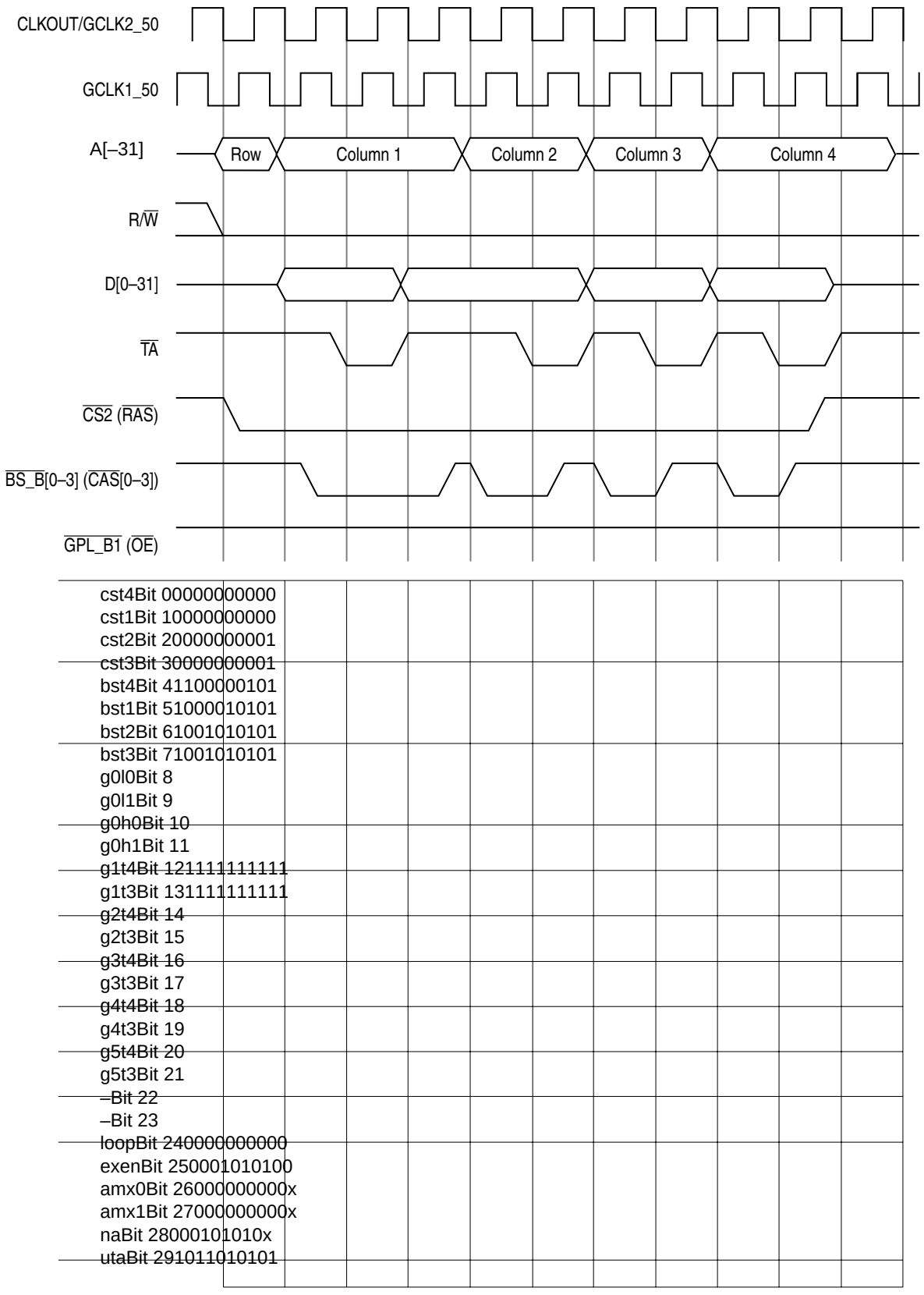


Figure 15-67. EDO DRAM Burst Write Access

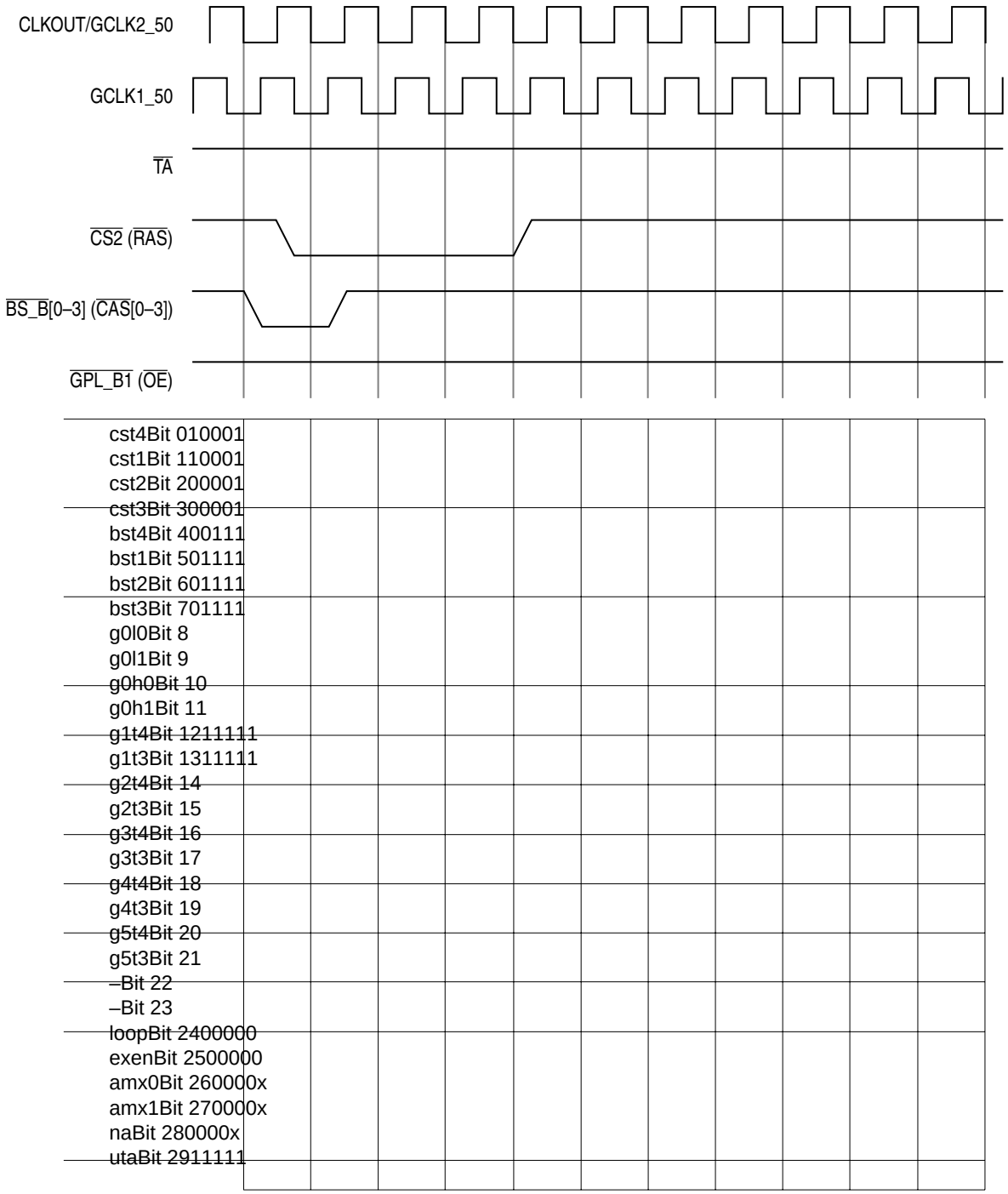


Figure 15-68. EDO DRAM Refresh Cycle (CAS before RAS)

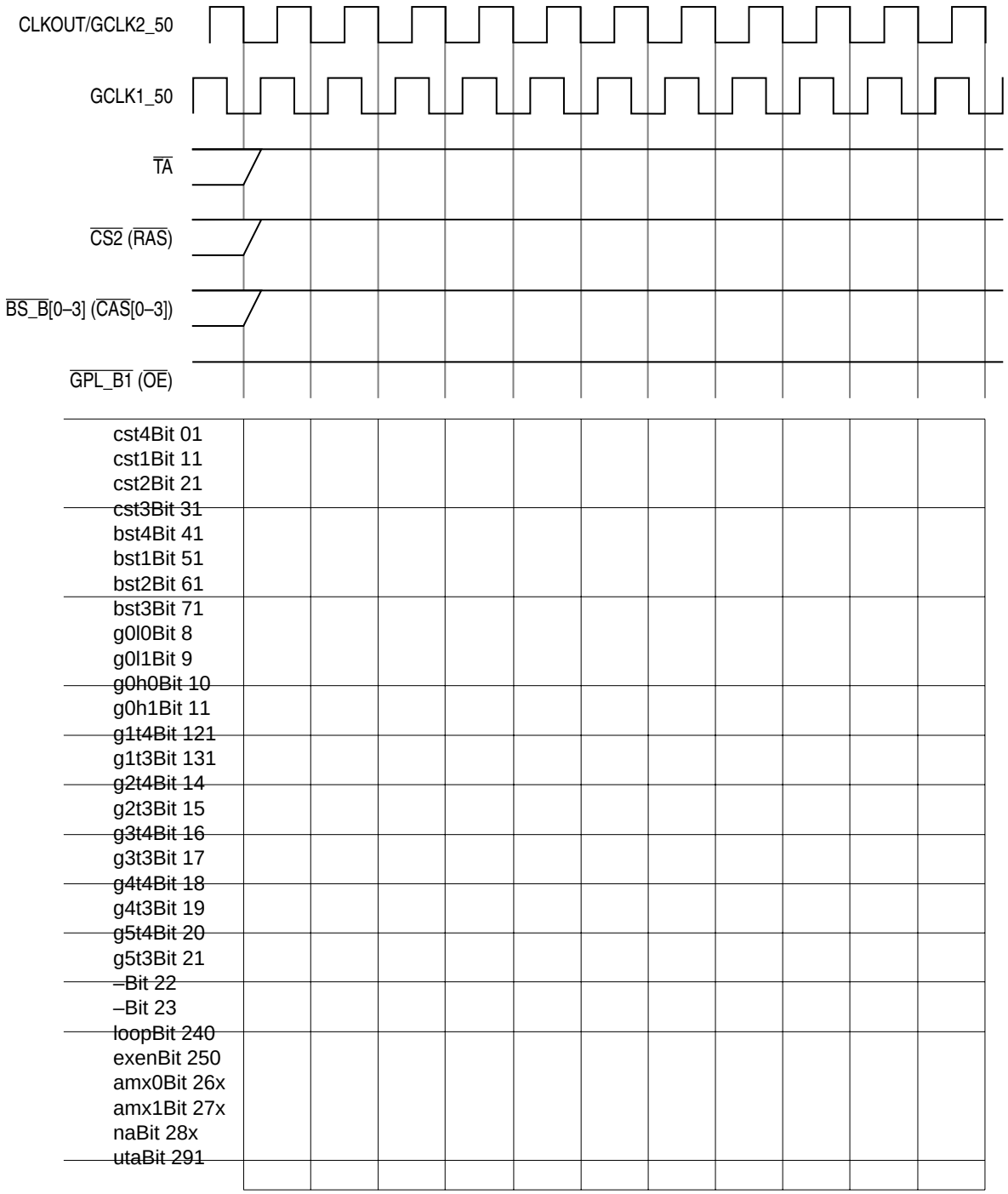


Figure 15-69. EDO DRAM Exception Cycle

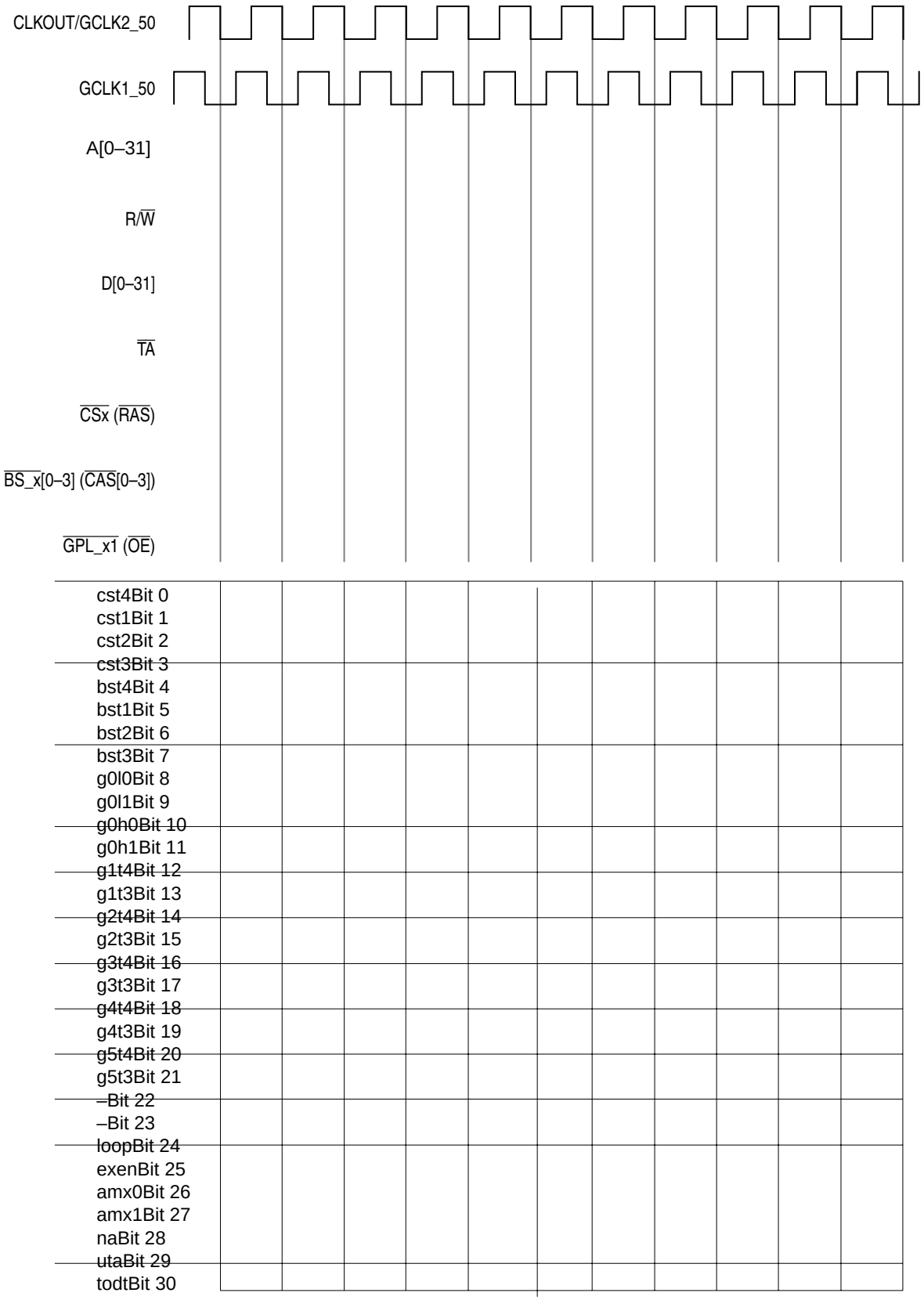


Figure 15-70. Blank Work Sheet for a UPM

Chapter 16
PCMCIA Interface

The PCMCIA host adapter module provides all control logic for a PCMCIA socket interface, and requires only additional external analog power switching logic and buffering. Additional external buffers allow the PCMCIA host adapter module to support up to two PCMCIA sockets. The PCMCIA interface supports the following:

- Host adapter interface fully compliant with the PCMCIA standard, release 2.1+ (PC Card -16).
- Up to two PCMCIA sockets, requiring only external buffering and analog switching logic
- Eight memory or I/O windows that can be assigned to either socket

16.1 System Configuration

In this system configuration, the sockets and system bus must be electrically isolated using external buffers and bus transceivers. These buffers also provide voltage conversion required from the 3.3- to 5-V cards. These components should be powered by the card power supply. The PCMCIA host adapter provides the control signals for demultiplexing the signals shared among the sockets. Figure 16-1 shows a system configuration consisting of two PCMCIA sockets.

16.2 PCMCIA Module Signal Definitions

PCMCIA signals shared among all sockets consist of the address and data buses, socket control signals, and synchronous socket status signals. A[6–31] and D[0–15] are the address and data signals of the system bus. Figure 16-1 shows the PCMCIA host adapter module’s external signals.

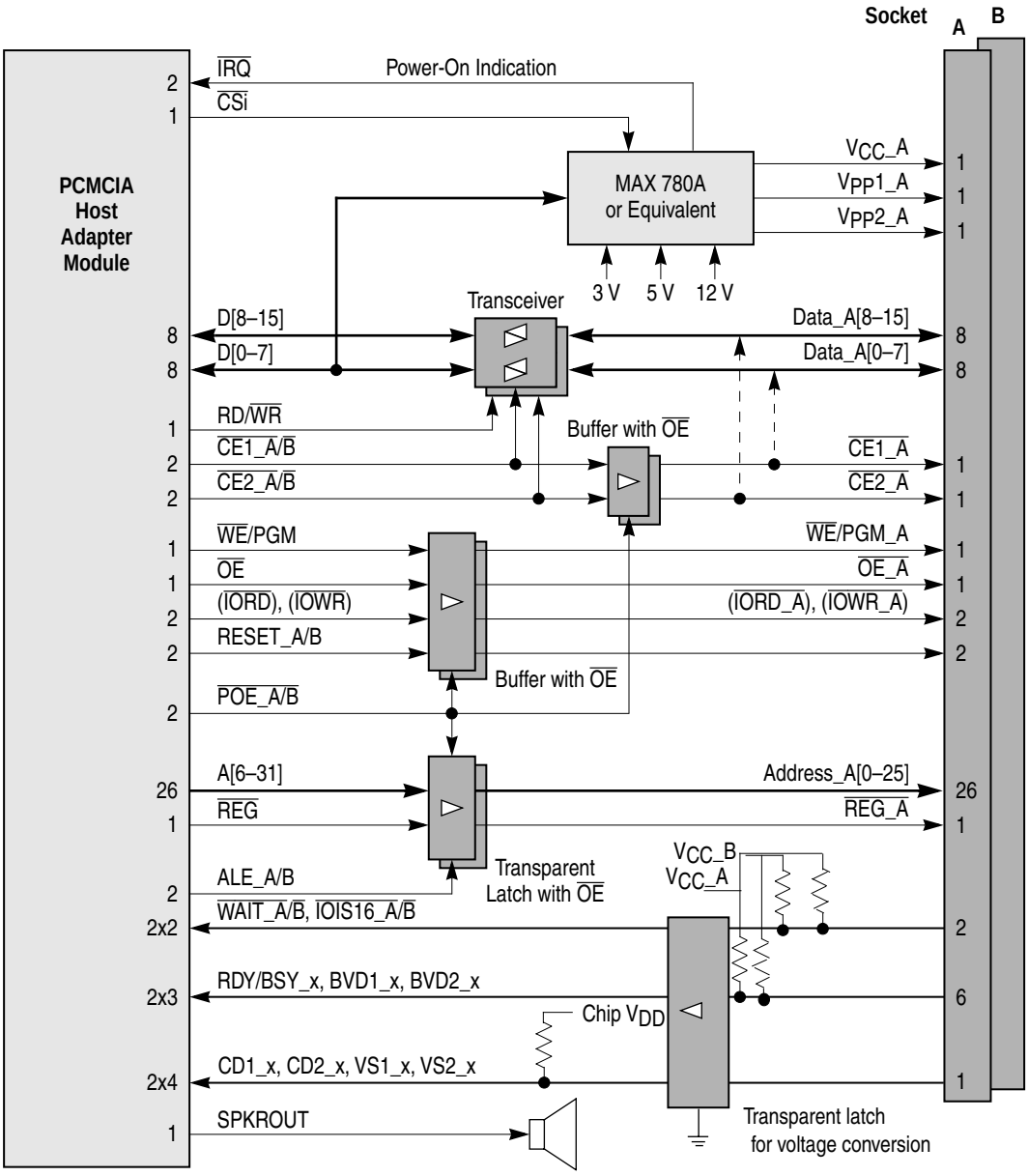


Figure 16-1. System with Two PCMCIA Sockets

16.2.1 PCMCIA Cycle Control Signals

Table 16-1 describes PCMCIA cycle control signals.

Table 16-1. PCMCIA Cycle Control Signals

<table>
<tr><th>Signal</th><th>Description</th></tr>
<tr><td>A[6–31]</td><td>Address bus. Output. A[6–31] should be buffered to generate the socket signals A[25–0]. These address bus output lines allow direct addressing of 64 Mbytes of memory on each PCMCIA card. A6 is the msb.</td></tr>
<tr><td>$\overline{\text{REG}}$</td><td>Attribute memory select. Output. When $\overline{\text{REG}}$ is asserted during a PCMCIA access, card access is limited to attribute memory when a memory access occurs ($\overline{\text{WE}}$ or $\overline{\text{OE}}$ are asserted) and to I/O ports when an I/O access occurs ($\overline{\text{IORD}}$ or $\overline{\text{IOWR}}$ are asserted). If $\overline{\text{REG}}$ is asserted, accesses to common memory or DMA devices are blocked. When no PCMCIA access is performed, this signal is TSIZ0.</td></tr>
<tr><td>$\overline{\text{CE1\_x}}$, $\overline{\text{CE2\_x}}$</td><td>Card enable. Output. When a PCMCIA access is performed, $\overline{\text{CE1}}$ enables even bytes; $\overline{\text{CE2}}$ enables odd bytes, as shown below.
<table>
<tr><th>Port Size</th><th>Access Size</th><th>MPC855T:A31 (Slot: A0)</th><th>$\overline{\text{CE2}}$</th><th>$\overline{\text{CE1}}$</th></tr>
<tr><td rowspan="3">8 bits</td><td>16-bit (even only)</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>8-bit odd</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>8-bit even</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td rowspan="3">16 bits</td><td>16-bit (even only)</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>8-bit odd</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>8-bit even</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td colspan="2">No access</td><td>X</td><td>1</td><td>1</td></tr>
</table></td></tr>
<tr><td>D[0–15]</td><td>Data bus. Bidirectional. D[0–15] constitute the bidirectional data bus. The msb is D0 and the lsb is D15.</td></tr>
<tr><td>$\overline{\text{WAIT\_x}}$</td><td>Extend bus cycle. Input. Asserted by the PC card to delay completion of the pending memory or I/O cycle.</td></tr>
<tr><td>RD/$\overline{\text{WR}}$</td><td>External transceiver direction. Output. Asserted during MPC855T read cycles and negated during write cycles. Used in the PCMCIA interface to control the direction of the data bus transceivers.</td></tr>
<tr><td>$\overline{\text{IORD\_x}}$</td><td>I/O read. Output. During PCMCIA accesses, this signal is asserted together with $\overline{\text{REG\_x}}$ and is used to read data from the PC card I/O space. $\overline{\text{IORD\_x}}$ is valid only when the $\overline{\text{REG\_x}}$ and at least one of the $\overline{\text{CE1\_x}}$ and $\overline{\text{CE2\_x}}$ signals is also asserted.</td></tr>
<tr><td>$\overline{\text{IOWR\_x}}$</td><td>I/O write. Output. Asserted with $\overline{\text{REG\_x}}$ during PCMCIA accesses used to latch data into the PC card I/O space. Valid only when $\overline{\text{REGx}}$ and either or both $\overline{\text{CE1\_x}}$ and $\overline{\text{CE2\_x}}$ signals are also asserted.</td></tr>
<tr><td>$\overline{\text{OE\_x}}$</td><td>Output enable. Output. During PCMCIA accesses, $\overline{\text{OE\_x}}$ is used to drive memory read data from a PC card in a PCMCIA socket.</td></tr>
<tr><td>$\overline{\text{WE\_x}}$</td><td>Write enable/program. Output. During PCMCIA accesses, $\overline{\text{WE\_x}}$ is used to latch memory write data to the PC card in a PCMCIA socket. Can also be used as the programming strobe for PC cards using programmable memory technologies.</td></tr>
</table>

Table 16-1. PCMCIA Cycle Control Signals (continued)

Signal	Description
ALE_x	Address latch enable. Output strobes that control the external latches of the address and $\overline{\text{REG}}$ signals for the appropriate PC card being accessed. ALE_A is asserted when socket A is accessed and ALE_B is asserted when socket B is accessed. Note that latches are used when power consumption is an issue. They keep the PCMCIA signals from toggling unnecessarily when the PCMCIA cards are not being accessed. If power consumption is not an issue, buffers can be used instead of ALE_x signals.
$\overline{\text{IOIS16\_x}}$	I/O port is 16 bits. Input. Applies only when the card and its socket are programmed for I/O interface operation. 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 supports 16-bit accesses. If the I/O region in which the address resides is programmed as 8 bits wide, $\overline{\text{IOIS16\_x}}$ is ignored.

16.2.2 PCMCIA Input Port Signals

The following signals are used by a PCMCIA slot to indicate card status. The MPC855T provides synchronization, transition detection, optional interrupt generation, and the means for the software to read the signal state. This function is not necessarily specific to PCMCIA; a system can use these signals as a general-purpose input port with edge detection and interrupt capability. These signals appear on pins IP_A[0–7] and IP_B[0–7]. All these signals are symmetrical except IP_x7, which have extended edge detection capability and IP_x2, which serve as an $\overline{\text{IOIS16\_x}}$ cycle-control signals for PCMCIA cycles.

Table 16-2. PCMCIA Input Port Signals

Signal	Description
$\overline{\text{VS1\_x}}$, $\overline{\text{VS2\_x}}$	Voltage sense. Input. Used as VS1 and VS2 and generated by PC cards. They notify the socket of the card V_{CC} requirement. These signals are connected to IP_x[0–1].
WP	Write protect. Input. When the card and socket are programmed for memory interface operation, this signal is used as WP. It reflects the state of the write-protect switch on the PC card. The PC card must assert WP when the card switch is enabled. It must be negated when the switch is disabled. For a PC card that is writable without a switch, WP must be connected to ground. If the PC card is permanently write-protected, WP must be connected to V_{CC}. These signals are connected to IP_x2 pins.
$\overline{\text{CD1\_x}}$, $\overline{\text{CD2\_x}}$	Card detect. Input. Provide proper detection of card insertion. They must be connected to ground internally on the PC card, thus, these signals are forced low when 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 IP_x4 and IP_x3, respectively.
BVD1_x, BVD2_x	Battery voltage detect. Input. When the card and its socket are programmed for memory interface operation, these signals are used as BVD1_x and BVD2_x and are generated by PC cards with on-board batteries to report the battery condition. Both BVD1_x and BVD2_x must be held asserted when the battery is in good condition. Negating BVD2_x while keeping BVD1_x asserted indicates the battery is in a warning condition and should be replaced, although data integrity on the card is still assured. Negating BVD1_x indicates that the battery is no longer serviceable and data is lost, regardless of the state of BVD2_x. These signals are connected to IP_x6 and IP_x5, respectively.
$\overline{\text{STSCHG\_x}}$	Status change. Input. When the card and its socket are programmed for I/O interface operation, BVD1_x is used as $\overline{\text{STSCHG\_x}}$ and is generated by I/O PC cards. $\overline{\text{STSCHG\_x}}$ must be held negated when the “signal on change” bit and “changed” bit in the card status register on the PC card are either or both zero. $\overline{\text{STSCHG\_x}}$ must be asserted when both bits = 1.

Table 16-2. PCMCIA Input Port Signals (continued)

Signal	Description
SPKR_x	Speaker. input. When the card and socket are programmed for I/O interface operation, BVD2_x is used as digital audio (SPKR_x) and is generated by I/O PC cards. SPKR_x must be used to provide the socket's single amplitude (digital) audio wave form to the system. The SPKR_x signal of all sockets are XORed and routed through the speaker out signal (SPKROUT).
RDY/ BSY_x, IREQ_x	Ready/busy or interrupt request. Input. When the card and its socket are programmed for memory interface operation, this signal is used as RDY/BSY_x and must be asserted 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 interface operation, this signal is used as IREQ_x and must be asserted by a PC card to indicate that a device on the PC card requires service by host software. Must be held negated when no interrupt is requested. These signals are connected to IP_x7.

16.2.3 PCMCIA Output Port Signals (OP[0–4])

A PCMCIA slot can use the signals in Table 16-3 to control the RESET input and output enable of the buffers to the card. The MPC855T gives software a way to control the output signal state. This function is not necessarily specific to the PCMCIA interface; a system can use these signals as a general-purpose output port.

Table 16-3. PCMCIA Output Port Signals

Signal	Description
RESET_x	Card reset. Output. Provided to clear the card's configuration option register, thus placing the card in its default (memory-only interface) state and beginning an additional card initialization. RESET_A is connected to OP0 and RESET_B is connected to OP3.
POE_x	PCMCIA buffers output enable. An output port line reflecting the value of PGCRx[CxOE]. Used to three-state address and strobe lines addressing each slot. POE_A connects to OP1; POE_B connects to OP2.

16.2.4 Other PCMCIA Signals

The PCMCIA socket uses the signals in Table 16-4, although their function is not necessarily specific to PCMCIA.

Table 16-4. Other PCMCIA Signals

Signal	Description
IRQ	Power is on. Input. The card power supply circuitry can use two of the IRQ signals as general-purpose interrupt requests to notify the MPC855T when the card's power supply reaches the full required voltage.
SPKROUT	Speaker out. Output. Provides a digital audio wave form to be driven to the system's speaker. It is generated as a logic exclusive OR of the SPKR_A and SPKR_B input signals. Note: General purpose timer 1 can be used to drive SPKROUT. When enabled, timer 1 is exclusive ORed with the resulting exclusive OR of the SPKR_A and SPKR_B input signals to generate SPKROUT. See Section 17.2.2.6, "Timer 1 and SPKROUT."

16.3 Operation Description

This section describes the operation of memory and I/O cards, interrupt detection and handling, power control, and reset.

16.3.1 Memory-Only Cards

Table 16-5 lists worst-case conditions of host programming memory cards and assumes $\overline{\text{WAIT}}$ is not used. If it is, the minimum strobe time is at least 35 ns + 1 system clock.

Table 16-5. Host Programming for Memory Cards

Memory Access Time [1]	600 ns			200 ns			150 ns			100 ns		
	STP [2]	LNG [3]	HLD [4]	STP	LNG	HLD	STP	LNG	HLD	STP	LNG	HLD
Clock Cycle	**100**	**300**	**150**	**30**	**120**	**90**	**20**	**80**	**75**	**15**	**60**	**50**
20 ns (50 MHz)	6	24	8	2	8	5	2	6	4	1	4	3
30 ns (33.3 MHz)	4	16	5	2	5	3	1	4	3	1	3	2
40 ns (25 MHz)	3	12	4	1	4	3	1	3	2	1	2	2
62 ns (16 MHz)	2	8	3	1	2	2	1	2	2	1	1	1
83 ns (12 MHz)	2	6	2	1	2	2	1	1	1	1	1	1

1 Because the minimum hold time is one clock, the real access time is access time + one clock.
2 Worst-case setup time (STP). The worst-case setup time is address to strobe.
3 Length (LNG) is the minimum strobe time.
4 Worst-case hold time (HLD). The worst-case hold time is data disable from $\overline{\text{OE}}$.

16.3.2 I/O Cards

Table 16-6 lists worst-case conditions of host programming I/O cards.

Table 16-6. Host Programming For I/O Cards

Frequency	STP [1]	LNG	HLD
	60	**165**	**30**
20 ns (50 MHz)	4	8	2
30 ns (33.3 MHz)	3	6	1
40 ns (25 MHz)	3	4	1
62 ns (16 MHz)	2	3	1
83 ns (12 MHz)	2	2	1

1 Setup time worst-case is for a write. In these cases, setup=data_set_up_before_iord +1 clock.

16.3.3 Interrupts

The PCMCIA interface input pins register (PIPR) reports any changes on inputs from the PCMCIA card to the host (BVD, CD, RDY, VS). The contents of the PCMCIA interface status changed register (PSCR) are logically ANDed with the PCMCIA interface enable register (PER) to generate a PCMCIA interface interrupt. The interrupt level is user programmable and the PCMCIA interface can generate an additional interrupt for RDY/$\overline{\text{IRQ}}$ that can trigger on level (low or high) or edge (fall or rise) of the input signal.

16.3.4 Power Control

The user can perform a write cycle using one of the memory controller chip-select pins. This data includes the controls to the analog switch such as the MAXIM MAX780. However, no auto-power control is supported.

16.3.5 Reset and Three-State Control

The user can reset the PCMCIA cards or disable the output of the external latches by writing to PGCR*x*[C*x*RESET] and PGCR*x*[C*x*OE], respectively.

16.3.6 DMA

The MPC855T DMA module with the CPM microcode provides two independent DMA (IDMA) channels. See Section 19.3, “IDMA Emulation.” The PCMCIA module can be programmed to generate control for an I/O device implemented as a PCMCIA card to respond to DMA transfer. Any window can be programmed as a DMA window through POR*x*[PRS]. When configured appropriately, the PCMCIA controller supplies the required signals to the socket. Note that DMA to and from the PCMCIA interface is handled through dual-access DMA transfers.

DMA requests can be supplied through $\overline{\text{SPKR}}$_*x*, $\overline{\text{IOIS16}}$_*x*, or $\overline{\text{INPACK}}$. To support DMA, $\overline{\text{INPACK}}$ should be connected to $\overline{\text{DREQ0}}$ for slot A or to $\overline{\text{DREQ1}}$ for slot B. The source for a DMA request is programmed through PGCR*x*[C*x*DREQ]. If the internal DMA request is disabled, the DMA request is assumed to be $\overline{\text{DREQ0}}$/$\overline{\text{DREQ1}}$ and port C should assign PC15/14 as $\overline{\text{DREQ0}}$/$\overline{\text{DREQ1}}$. If the request is enabled, port C should not be programmed to be $\overline{\text{DREQ0}}$/$\overline{\text{DREQ1}}$.

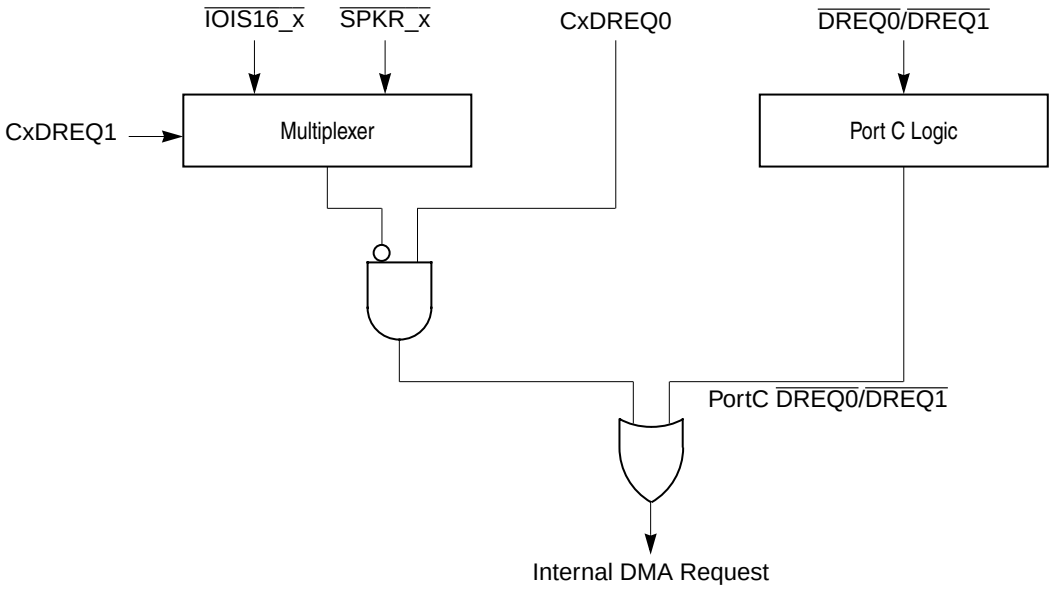


Figure 16-2. Internal DMA Request Logic

16.4 Programming Model

This section describes the PCMCIA interface programming model. Generally, all registers are memory-mapped within the internal control register area. The registers in Table 16-7 control the PCMCIA interface.

Table 16-7. PCMCIA Registers

Name	Description
PIPR	PCMCIA interface input pins register
PSCR	PCMCIA interface status changed register
PER	PCMCIA interface enable register
PGCRA	PCMCIA interface general control register a
PGCRB	PCMCIA interface general control register b
PBR[0–7]	PCMCIA base registers 0–7 (per window)
POR[0–7]	PCMCIA option registers 0–7 (per window)

16.4.1 PCMCIA Interface Input Pins Register (PIPR)

Status of inputs from the PCMCIA card to the host (BVD, CD, RDY, VS) is reported to the PIPR, shown in Figure 16-3. PIPR is a read-only register; write operations are ignored.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	CAVS1	CAVS2	CAWP	CACD2	CACD1	CABVD2	CABVD1	CARDY	—							
Reset	Undefined															
R/W	R															
Addr	(IMMR & 0xFFFF0000) + 0x0F0															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	CBVS1	CBVS2	CBWP	CBCD2	CBCD1	CBBVD2	CBBVD1	CBRDY	—							
Reset	Undefined															
R/W	R															
Addr	(IMMR & 0xFFFF0000) + 0x0F2															

Figure 16-3. PCMCIA Interface Input Pins Register (PIPR)

This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Table 16-8 describes PIPR fields.

Table 16-8. PIPR Field Descriptions

Bits	Name	Description
0	CAVS1	Voltage sense 1 for card A
1	CAVS2	Voltage sense 2 for card A
2	CAWP	Write protect for card A
3	CACD2	Card detect 2 for card A
4	CACD1	Card detect 1 for card A
5	CABVD2	Battery voltage/$\overline{\text{SPKR\_A}}$ input for card A
6	CABVD1	Battery voltage/$\overline{\text{STSCHG\_A}}$ input for card A
7	CARDY	RDY/$\overline{\text{IRQ}}$ of card A pin
8–15	—	Reserved, should be cleared.
16	CBVS1	Voltage sense 1 for card B
17	CBVS2	Voltage sense 2 for card B
18	CBWP	Write protect for card B
19	CBCD2	Card detect 2 for card B
20	CBCD1	Card detect 1 for card B
21	CBBVD2	Battery voltage/$\overline{\text{SPKR\_B}}$ input for card B
22	CBBVD1	Battery voltage/$\overline{\text{STSCHG\_B}}$ input for card B

Table 16-8. PIPR Field Descriptions (continued)

Bits	Name	Description
23	CBRDY	RDY/$\overline{\text{IRQ}}$ of card B pin
24–31	—	Reserved, should be cleared.

16.4.2 PCMCIA Interface Status Changed Register (PSCR)

The contents of PSCR, shown in Figure 16-4, are logically ANDed with the PER to generate a PCMCIA interface interrupt. Writing zeros has no effect; writing ones clears the corresponding interrupt state. This register is not affected by $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12–15
Field	CAVS1_C	CAVS2_C	CAWP_C	CACD2_C	CACD1_C	CABVD2_C	CABVD1_C	—	CARDY_L	CARDY_H	CARDY_R	CARDY_F	—
Reset	Undefined												
R/W	R/W												
Addr	(IMMR & 0xFFFF0000) + 0x0E8												
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28–31
Field	CBVS1_C	CBVS2_C	CBWP_C	CBCD2_C	CBCD1_C	CBBVD2_C	CBBVD1_C	—	CBRDY_L	CBRDY_H	CBRDY_R	CBRDY_F	—
Reset	Undefined												
R/W	R/W												
Addr	(IMMR & 0xFFFF0000) + 0x0EA												

Figure 16-4. PCMCIA Interface Status Changed Register (PSCR)

Table 16-9 describes PSCR fields.

Table 16-9. PSCR Field Descriptions

Bits	Name	Description
0	CAVS1_C	Voltage sense 1 for card A changed
1	CAVS2_C	Voltage sense 2 for card A changed
2	CAWP_C	Write protect for card A changed
3	CACD2_C	Card detect 2 for card A changed
4	CACD1_C	Card detect 1 for card A changed
5	CABVD2_C	Battery voltage/$\overline{\text{SPKR\_A}}$ input for card A changed
6	CABVD1_C	Battery voltage/$\overline{\text{STSCHG\_A}}$ input for card A changed
7	—	Reserved, should be cleared.
8	CARDY_L	RDY/$\overline{\text{IRQ}}$ of card A pin is low. Device and socket interrupt.
9	CARDY_H	RDY/$\overline{\text{IRQ}}$ of card A pin is high. Device and socket interrupt.
10	CARDY_R	RDY/$\overline{\text{IRQ}}$ of card A pin rising edge detected. Device and socket interrupt.

Table 16-9. PSCR Field Descriptions (continued)

Bits	Name	Description
11	CARDY_F	RDY/$\overline{\text{IRQ}}$ of card A pin falling edge detected. Device and socket interrupt.
12–15	—	Reserved, should be cleared.
16	CBVS1_C	Voltage sense 1 for card B changed
17	CBVS2_C	Voltage sense 2 for card B changed
18	CBWP_C	Write Protect for card B changed
19	CBCD2_C	Card detect 2 for card B changed
20	CBCD1_C	Card detect 1 for card B changed
21	CBBVD2_C	Battery voltage/$\overline{\text{SPKR\_B}}$ input for card B changed
22	CBBVD1_C	Battery voltage/$\overline{\text{STSCHG\_B}}$ input for card B changed
23	—	Reserved, should be cleared.
24	CBRDY_L	RDY/$\overline{\text{IRQ}}$ of card B pin is low. Device and socket interrupt.
25	CBRDY_H	RDY/$\overline{\text{IRQ}}$ of card B pin is high. Device and socket interrupt.
26	CBRDY_R	RDY/$\overline{\text{IRQ}}$ of card B pin rising edge detected. Device and socket interrupt.
27	CBRDY_F	RDY/$\overline{\text{IRQ}}$ of card B pin falling edge detected. Device and socket interrupt.
28–31	—	Reserved, should be cleared.

16.4.3 PCMCIA Interface Enable Register (PER)

Setting a bit in the PER, shown in Figure 16-5, enables the corresponding interrupt.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12-15
Field	CA_EVS1	CA_EVS2	CA_EWP	CA_ECD2	CA_ECD1	CA_EBVD2	CA_EBVD1	—	CA_ERDY_L	CA_ERDY_H	CA_ERDY_R	CA_ERDY_F	—
Reset	0000_0000_0000_0000												
R/W	R/W												
Addr	(IMMR & 0xFFFF0000) + 0x0F8												
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28-31
Field	CB_EVS1	CB_EVS2	CB_EWP	CB_ECD2	CB_ECD1	CB_EBVD2	CB_EBVD1	—	CB_ERDY_L	CB_ERDY_H	CB_ERDY_R	CB_ERDY_F	—
Reset	0000_0000_0000_0000												
R/W	R/W												
Addr	(IMMR & 0xFFFF0000) + 0x0FA												

Figure 16-5. PCMCIA Interface Enable Register (PER)

This register is affected by HRESET and SRESET. Table 16-10 describes PER fields.

Table 16-10. PER Field Descriptions

Bits	Name	Description
0	CA_EVS1	Enable for voltage sense 1 for card A changed. Setting this bit enables the interrupt on any signal change.
1	CA_EVS2	Enable for voltage sense 2 for card A changed. Setting this bit enables the interrupt on any signal change.
2	CA_EWP	Enable for write protect for card A changed. Setting this bit enables the interrupt on any signal change.
3	CA_ECD2	Enable for card detect 2 for card A changed. Setting this bit enables the interrupt on any signal change.
4	CA_ECD1	Enable for card detect 1 for card A changed. Setting this bit enables the interrupt on any signal change.
5	CA_EBVD2	Enable for battery voltage/SPKR_A input for card A changed. Setting this bit enables the interrupt on any signal change.
6	CA_EBVD1	Enable for battery voltage/STSCHG_A input for card A changed.
7	—	Reserved, should be 0.
8	CA_ERDY_L	Enable for RDY/IRQ of card A pin is low
9	CA_ERDY_H	Enable for RDY/IRQ card A pin is high
10	CA_ERDY_R	Enable for RDY/IRQ card A pin rising edge detected
11	CA_ERDY_F	Enable for RDY/IRQ card A pin falling edge detected
12–15	—	Reserved, should be 0.
16	CB_EVS1	Enable for voltage sense 1 for card B changed. Setting this bit enables the interrupt on any signal change.
17	CB_EVS2	Enable for voltage sense 2 for card B changed. Setting this bit enables the interrupt on any signal change.
18	CB_EWP	Enable for write protect for card B changed. Setting this bit enables the interrupt on any signal change.
19	CB_ECD2	Enable for card detect 2 for card B changed. Setting this bit enables the interrupt on any signal change.
20	CB_ECD1	Enable for card detect 1 for card B changed. Setting this bit enables the interrupt on any signal change.
21	CB_EBVD2	Enable for battery voltage/SPKR_B input for card B changed. Setting this bit enables the interrupt on any signal change.
22	CB_EBVD1	Enable for battery voltage/STSCHG_B input for card B changed
23	—	Reserved, should be 0.
24	CB_ERDY_L	Enable for RDY/IRQ of card B pin is low
25	CB_ERDY_H	Enable for RDY/IRQ card B pin is high
26	CB_ERDY_R	Enable for RDY/IRQ card B pin rising edge detected

Table 16-10. PER Field Descriptions (continued)

Bits	Name	Description
27	CB_ERDY_F	Enable for RDY/$\overline{\text{IRQ}}$ card B pin falling edge detected
28–31	—	Reserved, should be 0.

16.4.4 PCMCIA Interface General Control Register (PGCR*x*)

PGCRA or PGCRB, shown in Figure 16-6, are used to reset the PCMCIA cards, disable the output of the external latches, and specify the source used for a DMA request.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	CxIREQLVL								CxSCHLVL							
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	(IMMR & 0xFFFF0000) + 0x0E0 (PGCRA); 0x0E4 (PGCRB)															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	CxDREQ		—						CxOE	CxRESET	—					
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	(IMMR & 0xFFFF0000) + 0x0E2 (PGCRA); 0x0E6 (PGCRB)															

Figure 16-6. PCMCIA Interface General Control Register (PGCRx)

This register is affected by $\overline{\text{HRESET}}$ but is not affected by $\overline{\text{SRESET}}$. Table 16-11 describes PGCRx fields.

Table 16-11. PGCRx Field Descriptions

Bits	Name	Description
0–7	CxIREQLVL	Card x $\overline{\text{IREQ\_x}}$ interrupt level. Only one bit of this field should be set at any time.
8–15	CxSCHLVL	Card x $\overline{\text{STSCHG\_x}}$ interrupt level. Only one CASCHLVLx bit should be set at any time.
16–17	CxDREQ	Card x DREQ. Defines internal DMA request for the on-chip DMA controller (CADREQ controls DMA channel 0. CBDREQ controls DMA channel 1). 0x Disable internal DMA request from slot x. 10 Enable $\overline{\text{IOIS16\_x}}$ as internal DMA request for slot x. 11 Enable $\overline{\text{SPKR\_x}}$ as internal DMA request for slot x.
18–23	—	Reserved, should be cleared.
24	CxOE	Card x output enable. CAOE is reflected on OP1 and CBOE is reflected on OP2 used to three-state the external buffers when the card's power is activated.
25	CxRESET	Card x reset. CARESET is reflected on OP0 used to reset card A. CBRESET is reflected on OP3 used to reset card B.
26–31	—	Reserved, should be cleared.

16.4.5 PCMCIA Base Registers 0–7 (PBR0–PBR7)

Setting a bit in the PBR, shown in Figure 16-5, enables the corresponding interrupt.

Bit	0	1	2	3	4	5	…	31
Field	PBA							
Reset	—							
R/W	R/W							
Addr	(IMMR & 0xFFFF0000) + 0x080 (PBR0); 0x088 (PBR1); 0x090 (PBR2); 0x098 (PBR3); 0x0A0 (PBR4); 0x0A8 (PBR5); 0x0B0 (PBR6); 0x0B8 (PBR7)							

Figure 16-7. PCMCIA Base Register (PBR)

This register is not affected by $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$. Table 16-12 describes the PBR.

Table 16-12. PBR Field Descriptions

Bits	Name	Description
0–31	PBA	PCMCIA base address. Compared to the address on the address bus to determine if a PCMCIA window is being accessed by an internal bus master. PBA is used in conjunction with POR[BSIZE].

16.4.6 PCMCIA Option Register 0–7 (POR0–POR7)

The POR, shown in Figure 16-8, as the manipulation of timing, provides the address mask for the bank size, and defines the region, slot, write protection, and validation.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	BSIZE					—							PSHT			
Reset	Undefined															
R/W	R/W															
Addr	(IMMR & 0xFFFF0000) + 0x084 (POR0); 0x08C (POR1); 0x094 (POR2); 0x09C (POR3); 0x0A4 (POR4); 0x0AC (POR5); 0x0B4 (POR6); 0x0BC (POR7)															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	PSST				PSL					PPS	PRS			PSLOT	WP	PV
Reset	Undefined															
R/W	R/W															
Addr	(IMMR & 0xFFFF0000) + 0x086 (POR0); 0x08E (POR1); 0x096 (POR2); 0x09E (POR3); 0x0A6 (POR4); 0x0AE (POR5); 0x0B6 (POR6); 0x0BE (POR7)															

Figure 16-8. PCMCIA Option Register 0–7 (POR0–POR7)

This register is not affected by $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$. Table 16-13 describes POR fields.

Table 16-13. POR Field Descriptions

Bits	Name	Description
0–4	BSIZE	PCMCIA bank size. Determines the address mask field of each POR and provides masking for any of the corresponding bits in the associated PBR. The bank size is calculated as banksize = 2^{BSIZE}, where BSIZE represents the gray code shown below: 00000 1 byte 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 1 Kbyte 01110 2 Kbytes 01010 4 Kbytes 01011 8 Kbytes 01001 16 Kbytes 01000 32 Kbytes 11000 64 Kbytes 11001 128 Kbytes 11011 256 Kbytes 11010 512 Kbytes 11110 1 Mbyte 11111 2 Mbytes 11101 4 Mbytes 11100 8 Mbytes 10100 16 Mbytes 10101 32 Mbytes 10111 64 Mbytes
0–4	BSIZE (cont.)	BSIZE determines not only the bank size, but also how the address is compared with PBRB[PBA]. If virtual field, MASK, is defined as shown below: BSIZE MASK 00000 1111 1111 1111 1111 1111 1111 1111 1111 00001 1111 1111 1111 1111 1111 1111 1111 1110 00011 1111 1111 1111 1111 1111 1111 1111 1100 00010 1111 1111 1111 1111 1111 1111 1111 1000 00110 1111 1111 1111 1111 1111 1111 1111 0000 00111 1111 1111 1111 1111 1111 1111 1110 0000 00101 1111 1111 1111 1111 1111 1111 1100 0000 00100 1111 1111 1111 1111 1111 1111 1000 0000 01100 1111 1111 1111 1111 1111 1111 0000 0000 01101 1111 1111 1111 1111 1111 1110 0000 0000 01111 1111 1111 1111 1111 1111 1100 0000 0000 01110 1111 1111 1111 1111 1111 1000 0000 0000 01010 1111 1111 1111 1111 1111 0000 0000 0000 01011 1111 1111 1111 1111 1110 0000 0000 0000 01001 1111 1111 1111 1111 1100 0000 0000 0000 01000 1111 1111 1111 1111 1000 0000 0000 0000 11000 1111 1111 1111 1111 0000 0000 0000 0000 11001 1111 1111 1111 1110 0000 0000 0000 0000 11011 1111 1111 1111 1100 0000 0000 0000 0000 11010 1111 1111 1111 1000 0000 0000 0000 0000 11110 1111 1111 1111 0000 0000 0000 0000 0000 11111 1111 1111 1110 0000 0000 0000 0000 0000 11101 1111 1111 1100 0000 0000 0000 0000 0000 11100 1111 1111 1000 0000 0000 0000 0000 0000 10100 1111 1111 0000 0000 0000 0000 0000 0000 10101 1111 1110 0000 0000 0000 0000 0000 0000 10111 1111 1100 0000 0000 0000 0000 0000 0000 Addr & MASK = PBA & MASK for a valid PCMCIA access; otherwise, it is not a valid PCMCIA access
5–11	—	Reserved, should be cleared.

Table 16-13. POR Field Descriptions (continued)

Bits	Name	Description
12–15	PSHT	PCMCIA strobe hold time (strobe negation to address negation). Specifies when $\overline{\text{IOWR_x}}$x or $\overline{\text{WE\_x}}$ are negated during a PCMCIA write or when $\overline{\text{IORD\_x}}$ or $\overline{\text{OE\_x}}$ are negated during a PCMCIA read. Used to meet 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 ... 1111 Strobe negation to address change 15 clock
16–19	PSST	PCMCIA strobe set up time (address to strobe assertion). Specifies when $\overline{\text{IOWR\_x}}$ or $\overline{\text{WE\_x}}$ are asserted during a PCMCIA write access or when $\overline{\text{IORD\_x}}$ or $\overline{\text{OE\_x}}$ are asserted during a PCMCIA read access handled by the PCMCIA interface. This helps meet address/setup time requirements for slow memories and peripherals. 0000 Reserved 0001 Address to strobe assertion 1 clock cycle 0010 Address to strobe assertion 2 clock cycles ... 1111 Address to strobe assertion 15 clock cycles
20–24	PSL	PCMCIA strobe length. Determines the number of cycles the strobe is asserted during a PCMCIA access for this window and, thus, it is the main parameter for determining cycle length. The cycle may be lengthened by asserting $\overline{\text{WAIT}}$. 00001 Strobe asserted 1 clock cycles 00010 Strobe asserted 2 clock cycles ... 11111 Strobe asserted 31 clock cycles 00000 Strobe asserted 32 clock cycles
25	PPS	PCMCIA port size. Specifies the port size of this PCMCIA window. 0 8 bits port size 1 16 bits port size
26–28	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 11x Reserved Note: The DMA encoding generates a normal DMA transfer unless signaled as last by the on-chip DMA controller. In this case TC($\overline{\text{OE}}$) or TC ($\overline{\text{WE}}$) is asserted.The DMA last transaction encoding generates a DMA transfer with TC($\overline{\text{OE}}$) or TC ($\overline{\text{WE}}$) asserted, regardless of any internal indication.
29	PSLOT	PCMCIA slot identifier. 0 This window defined for slot A. 1 This window defined for slot B.
30	WP	Write-protect enable. 0 Not write protected. 1 Write protected. Attempting to write to this window causes a machine check interrupt.
31	PV	PCMCIA valid. Indicates whether the contents of the OBR and POR pair are valid. 0 This bank is invalid. 1 This bank is valid.

16.5 PCMCIA Controller Timing Examples

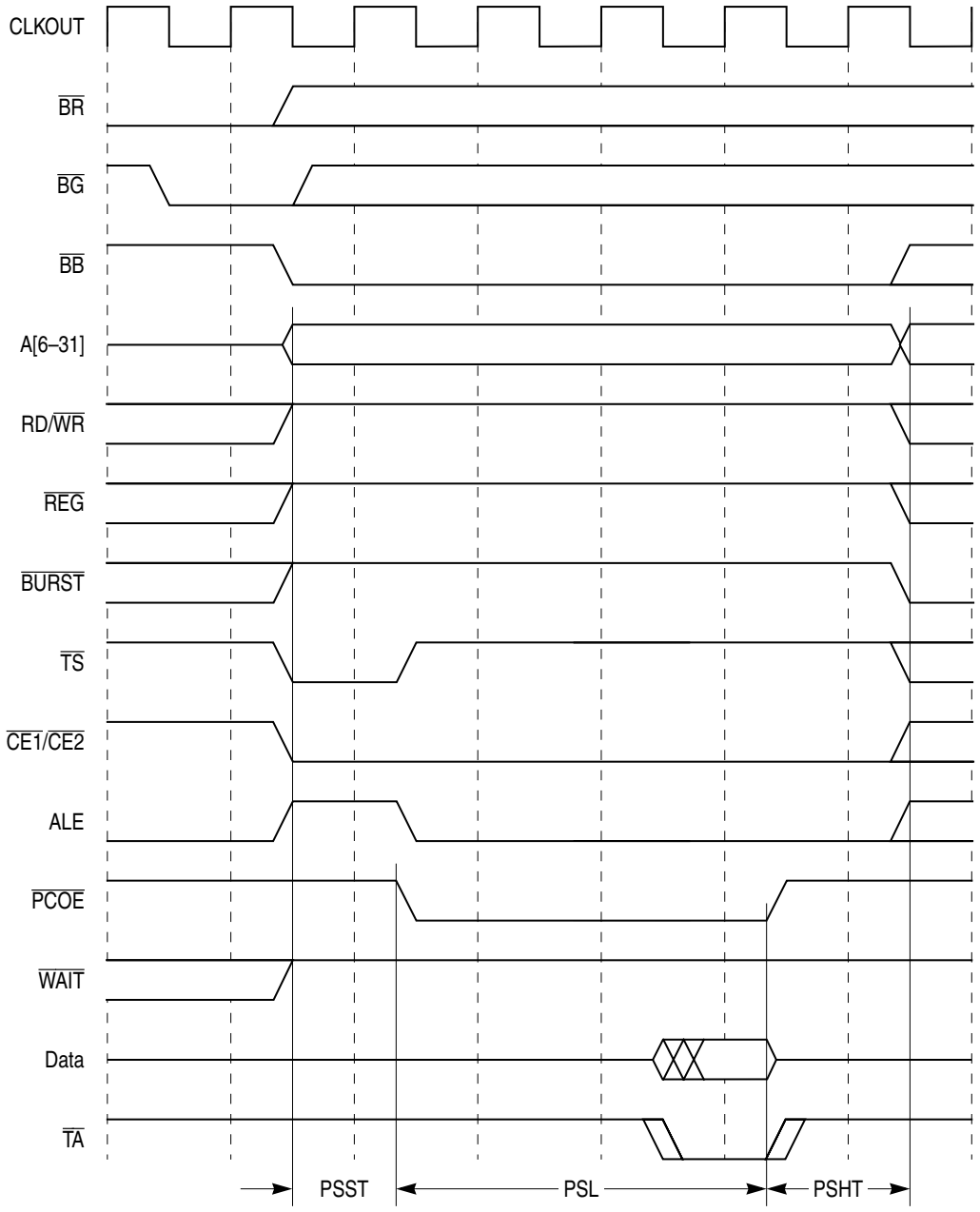


Figure 16-9. PCMCIA Single-Beat Read Cycle PRS = 0 PSST = 1 PSL = 3 PSHT = 1

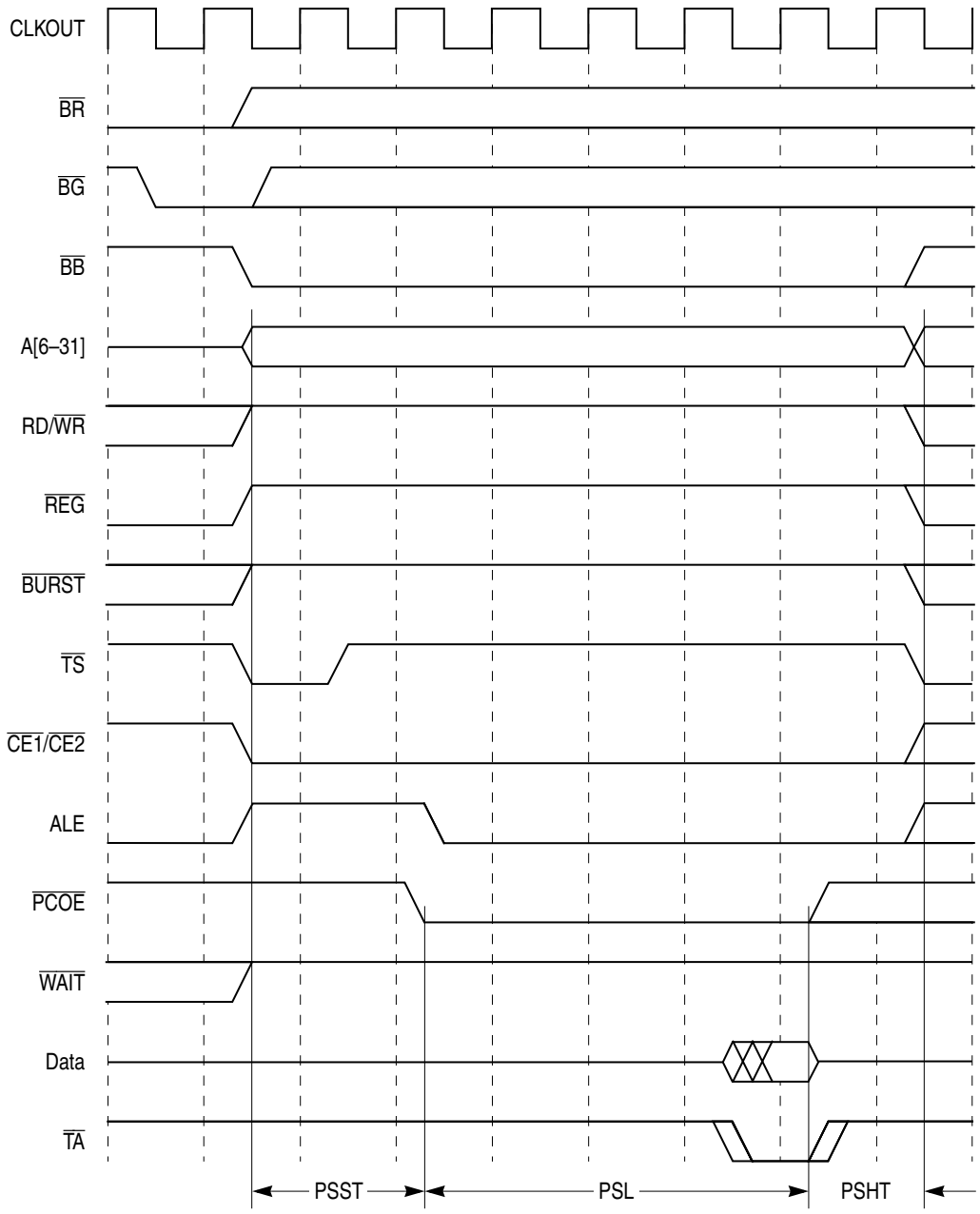


Figure 16-10. PCMCIA Single-Beat Read Cycle PRS = 0 PSST = 2 PSL = 4 PSHT = 1

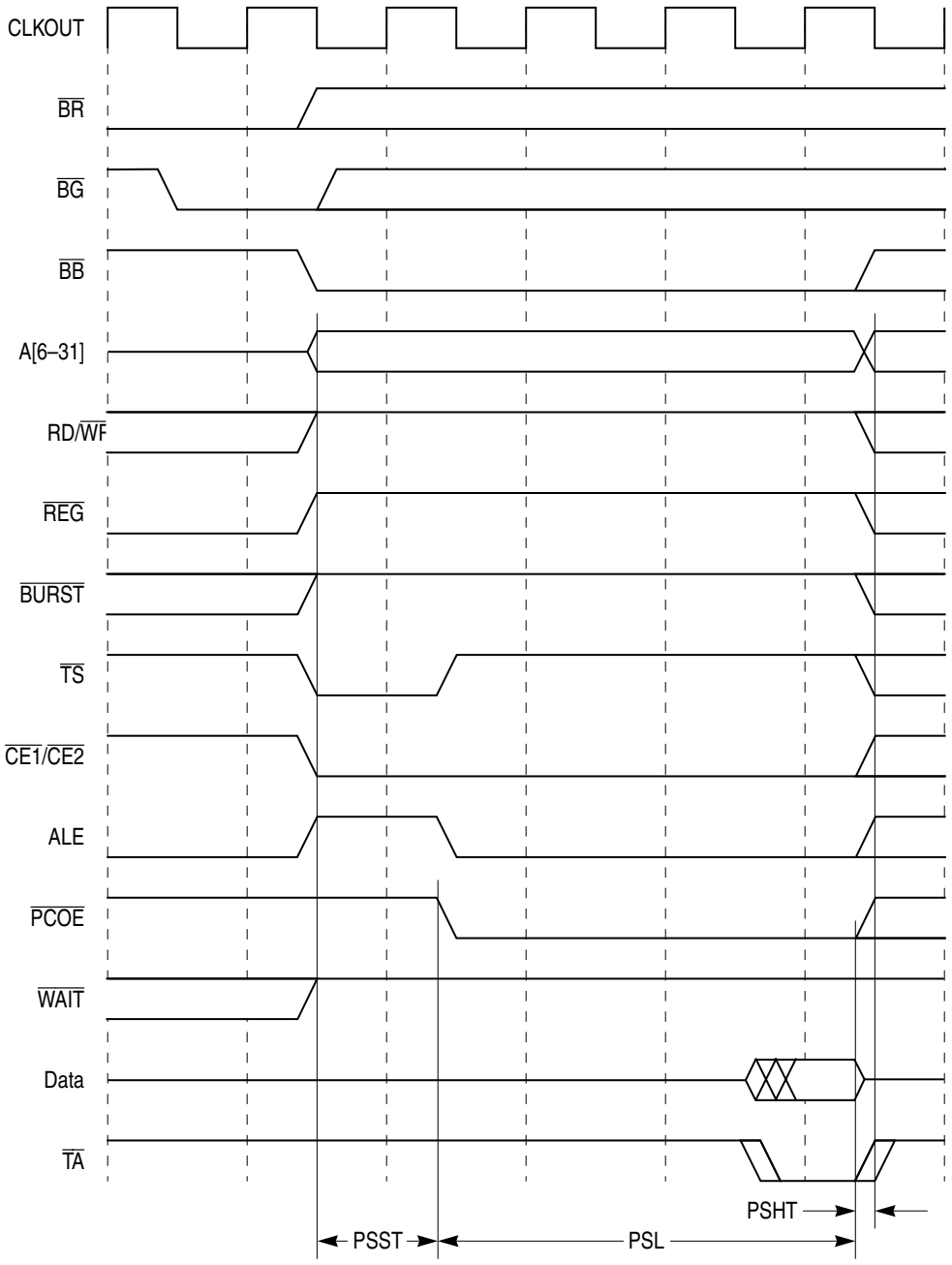


Figure 16-11. PCMCIA Single-Beat Read Cycle PRS = 0 PSST = 1 PSL = 3 PSHT = 0

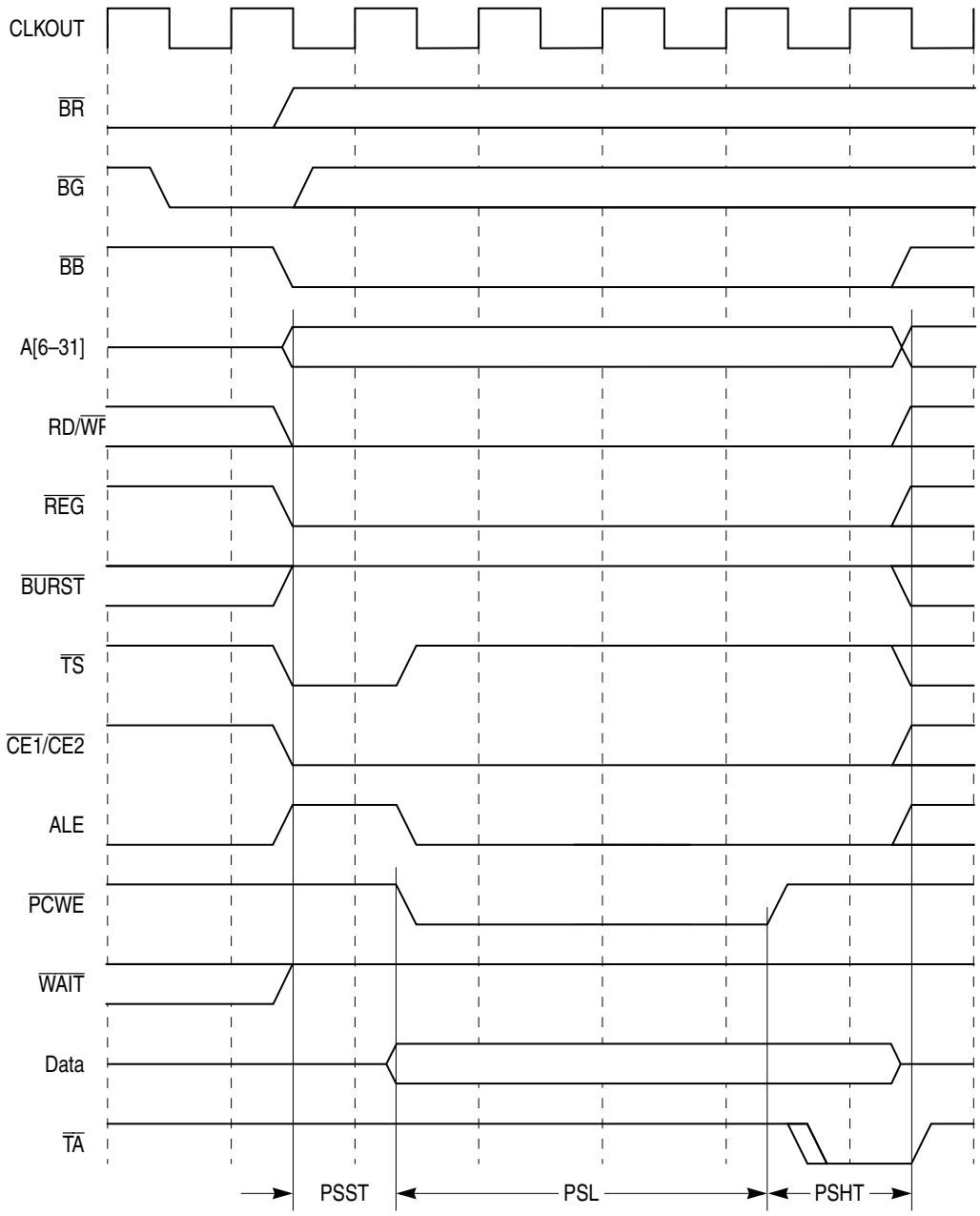


Figure 16-12. PCMCIA Single-Beat Write Cycle PRS = 2 PSST = 1 PSL = 3 PSHT = 1

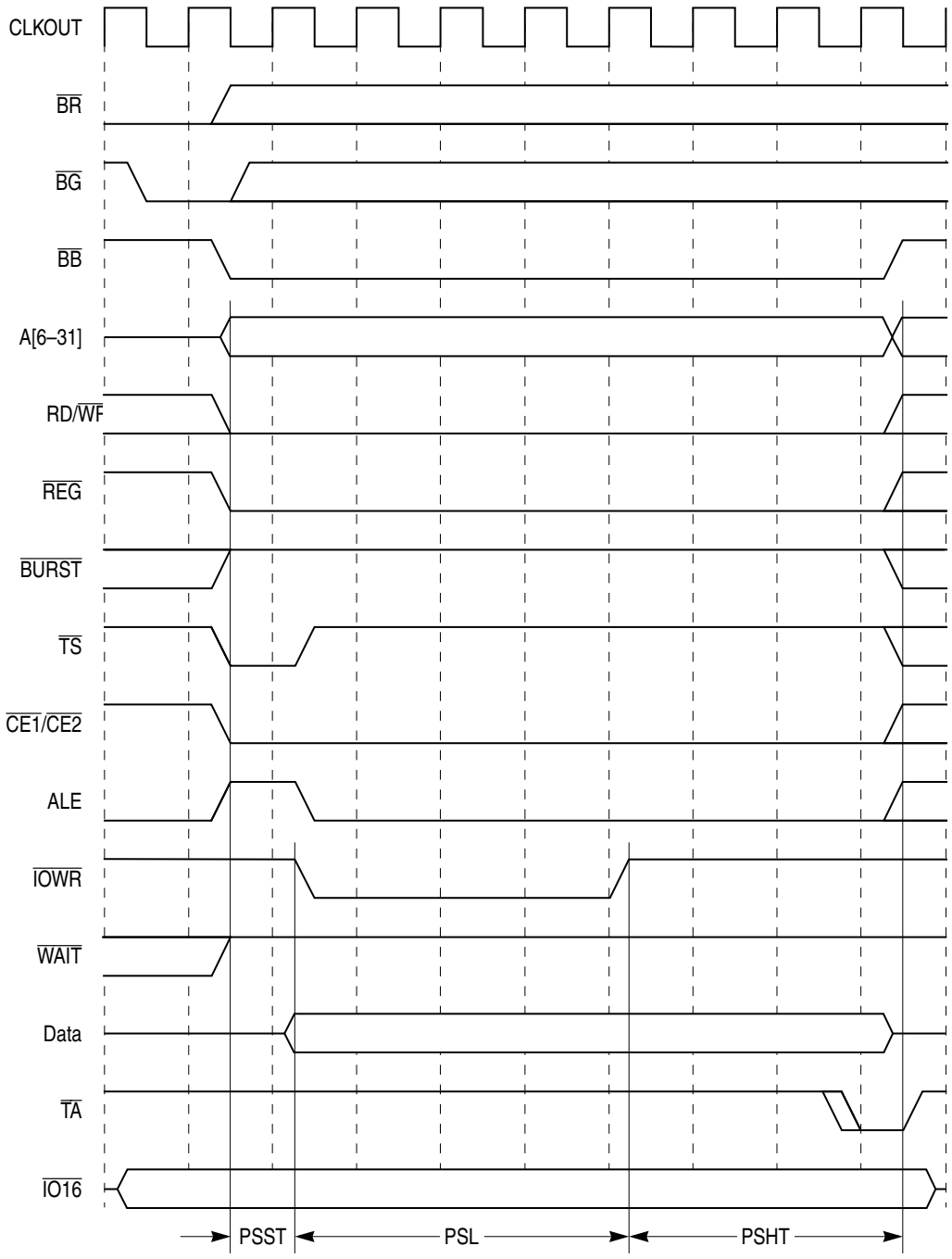


Figure 16-13. PCMCIA Single-Beat Write Cycle PRS = 3 PSST = 1 PSL = 4 PSHT = 3

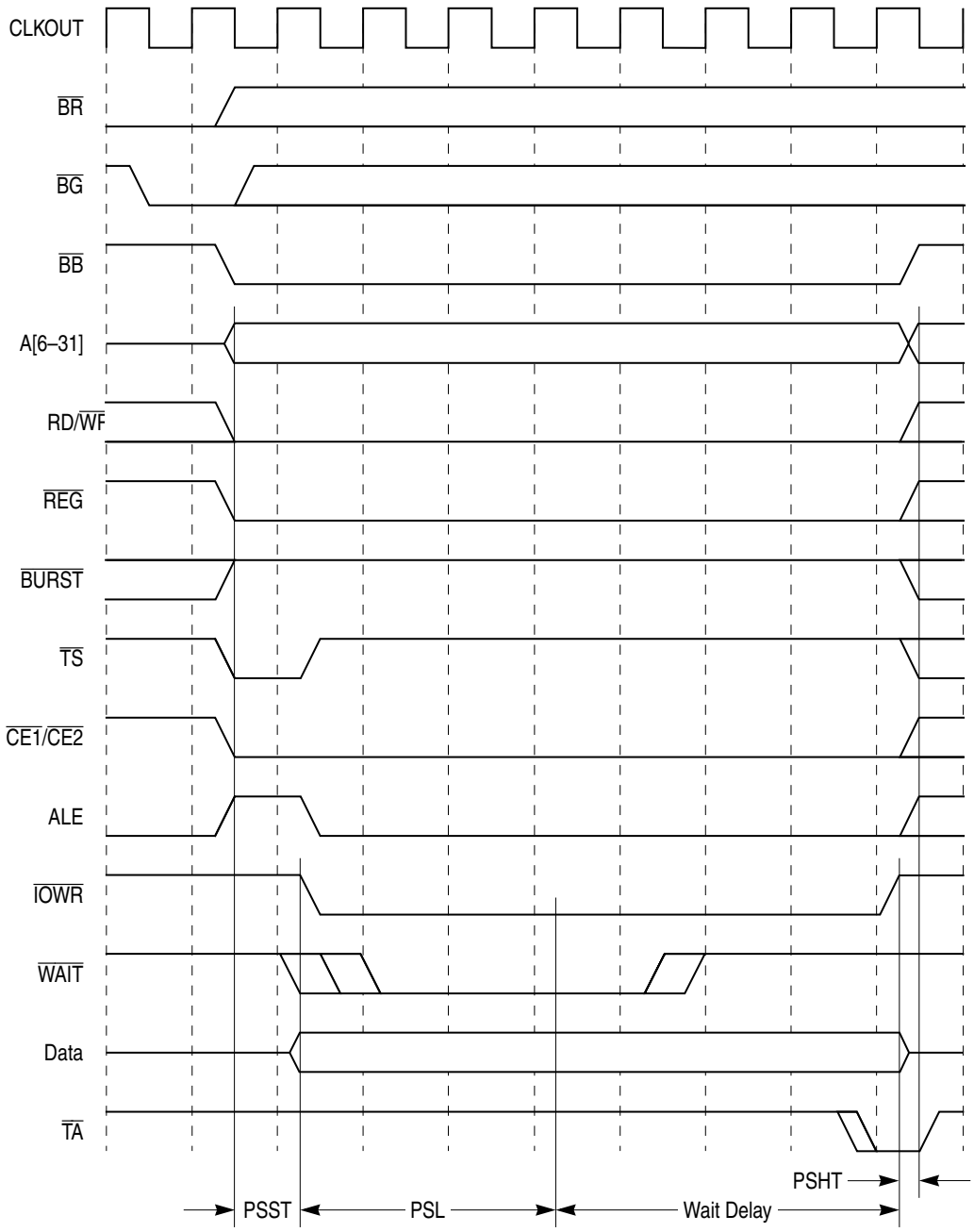


Figure 16-14. PCMCIA Single-Beat Write with Wait PRS = 3 PSST = 1 PSL = 3 PSHT = 0

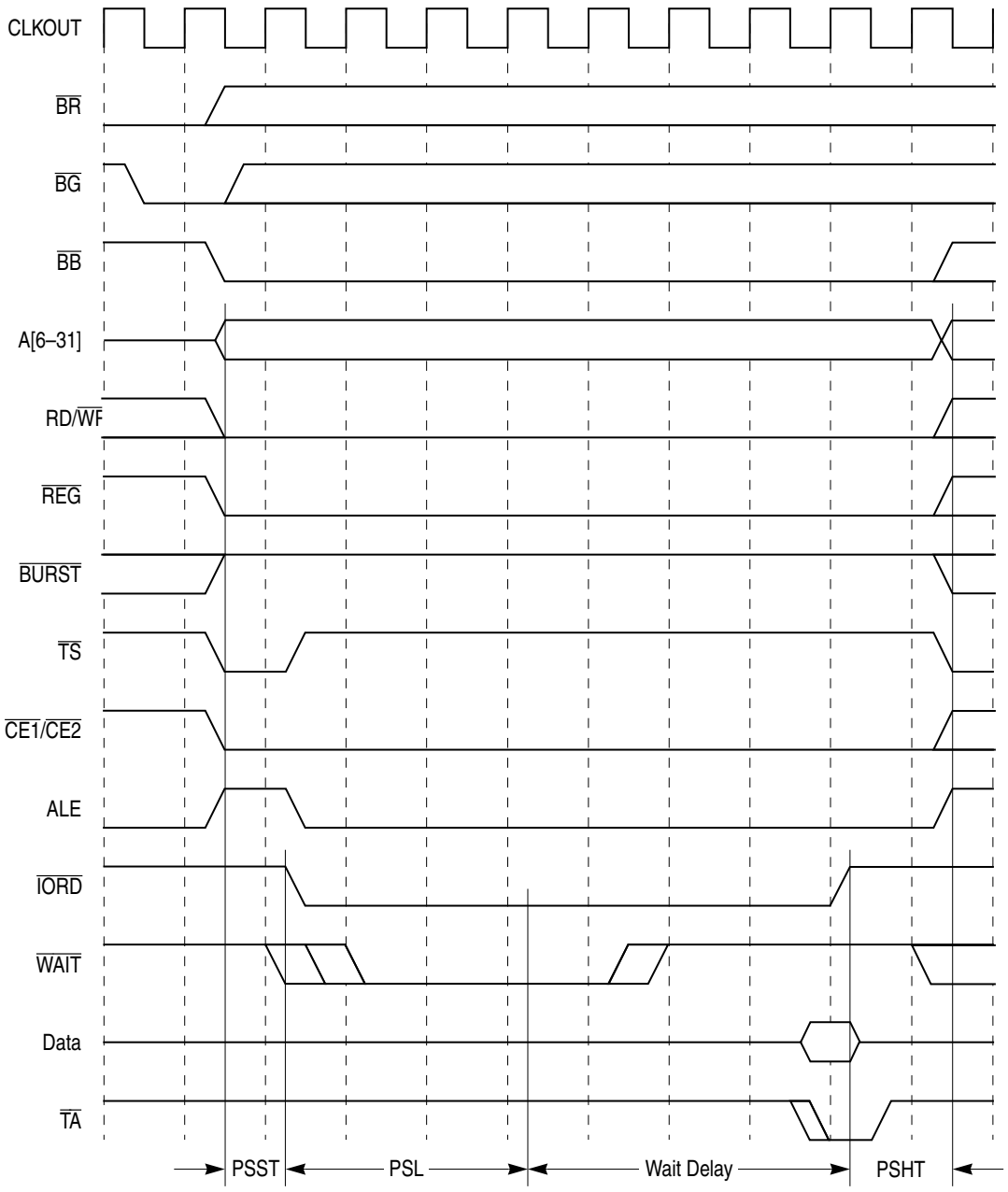


Figure 16-15. PCMCIA Single-Beat Read with Wait PRS = 3 PSST = 1 PSL = 3 PSHT =1

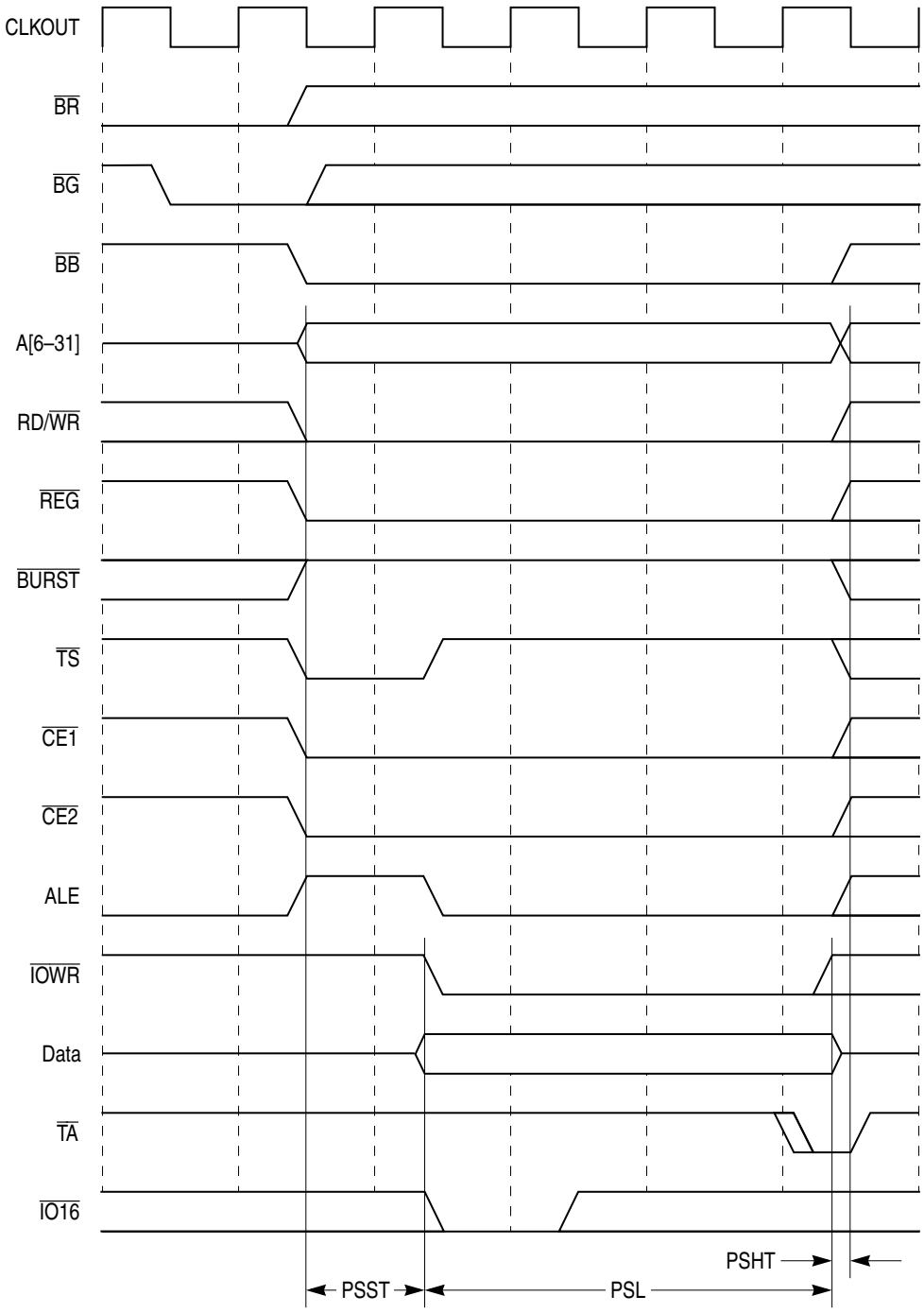


Figure 16-16. PCMCIA I/O Read PPS = 1 PRS = 3 PSST = 1 PSL = 2 PSHT = 0

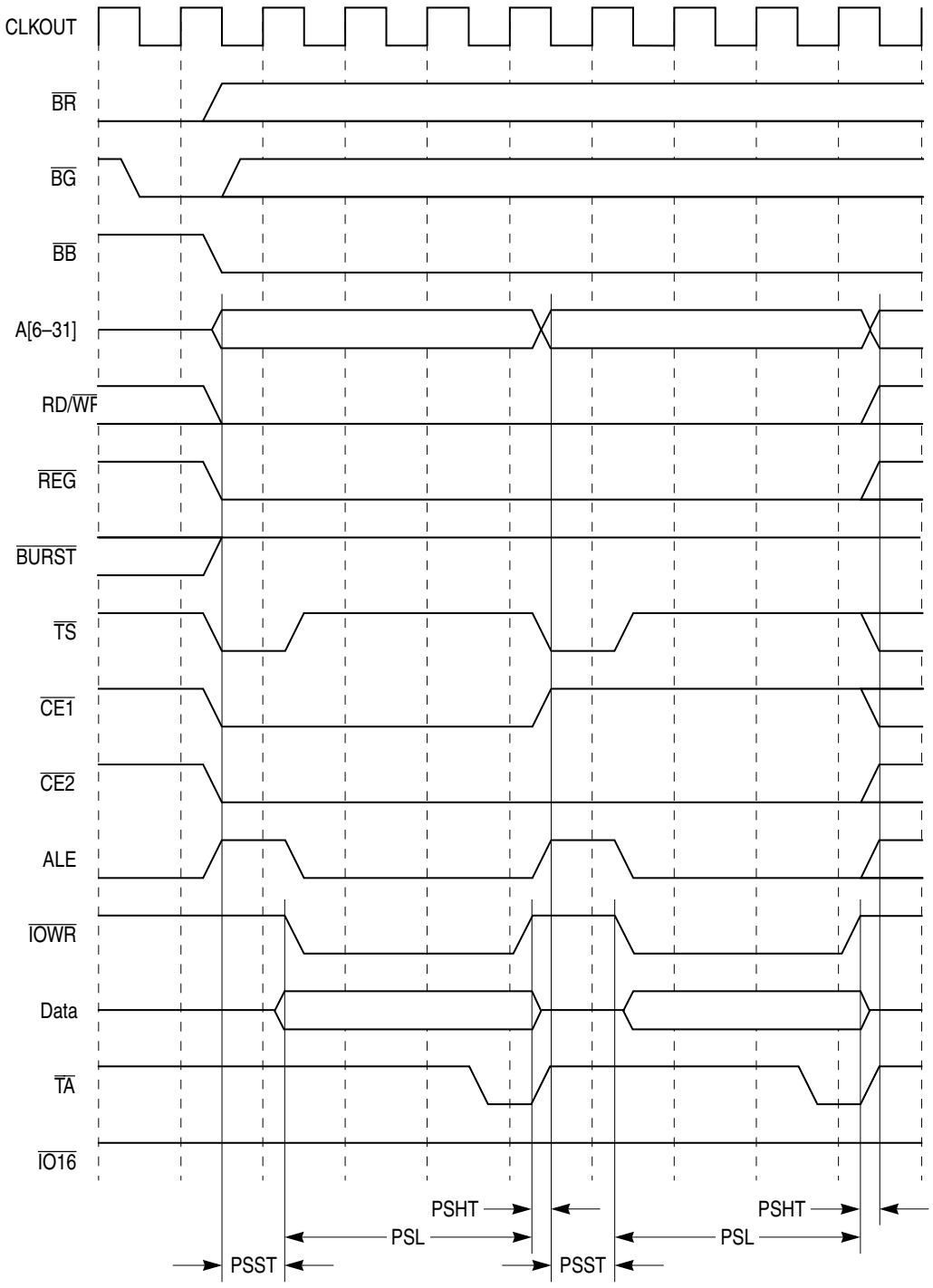


Figure 16-17. PCMCIA I/O Read PPS = 1 PRS = 3 PSST = 1 PSL = 2 PSHT = 0

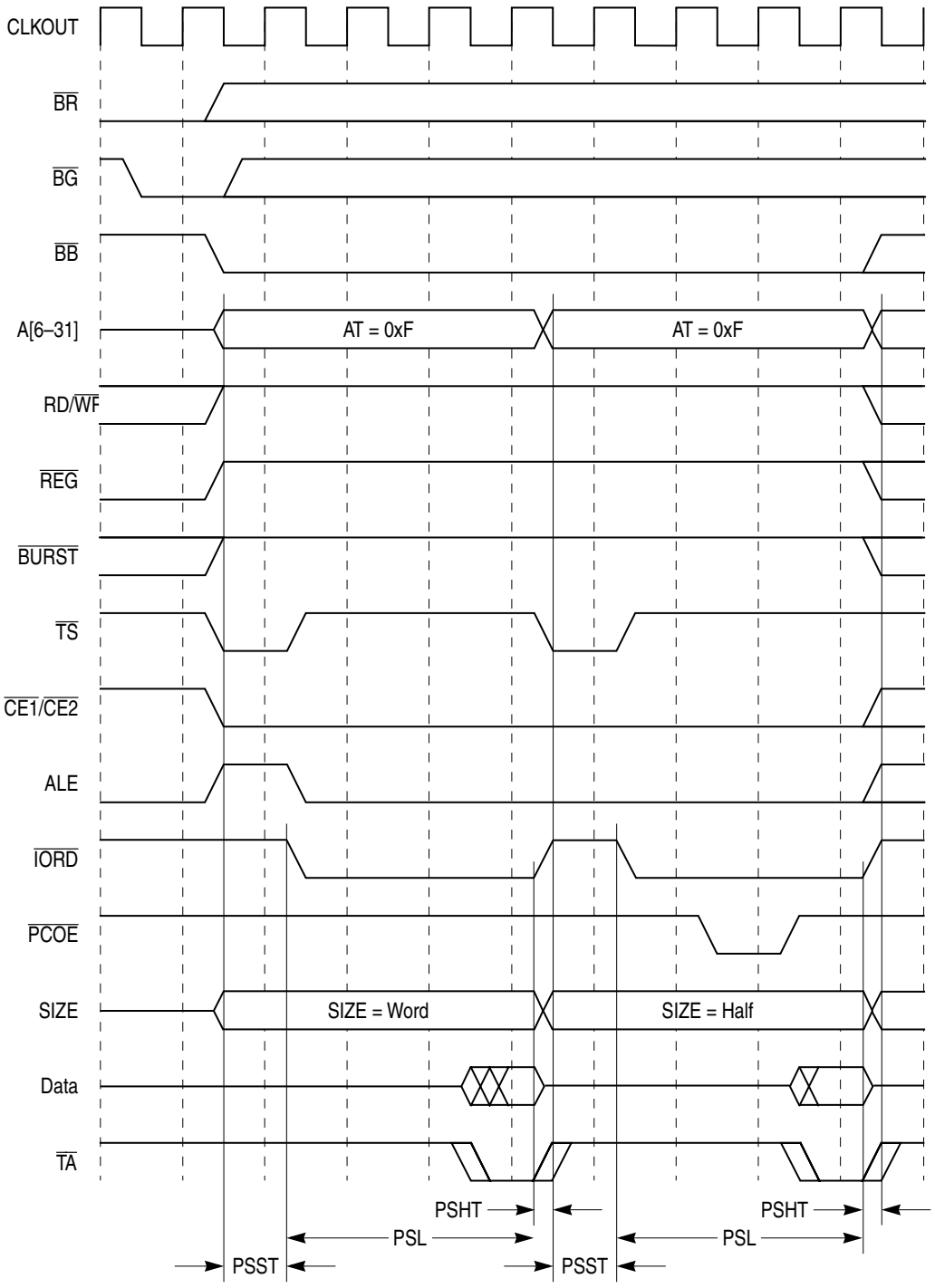


Figure 16-18. PCMCIA DMA Read Cycle PRS = 4 PSST = 1 PSL = 3 PSHT = 0

Part V
Communications Processor Module

Intended Audience

Part V is intended for system designers who need to implement various communications protocols on the MPC855T. It assumes a basic understanding of the PowerPC exception model, the MPC855T interrupt structure, as well as a working knowledge of the communications protocols to be used. A complete discussion of these protocols is beyond the scope of this book.

Contents

Part V describes behavior of the MPC855T communications processor module (CPM) and the RISC communications processor (CP) that it contains (note that this is separate from the embedded MPC8xx processor). It contains the following chapters:

- Chapter 17, “Communications Processor Module and CPM Timers,” provides a brief overview of the MPC855T CPM and a detailed discussion of the clocking mechanisms supported.
- Chapter 18, “Communications Processor,” describes the RISC communications processor (CP), which handles the low-level communications tasks, freeing the core for higher-level tasks.
- Chapter 19, “SDMA Channels and IDMA Emulation,” describes the two physical serial DMA (SDMA) channels on the MPC855T with which the CP implements virtual SDMA channels.
- Chapter 20, “Serial Interface,” describes the serial interface (SI) in which the physical interface to all SCCs and SMCs is implemented.
- Chapter 21, “Serial Communications Controller,” describes the serial communications controllers (SCC), which can be configured independently to implement different protocols for bridging functions, routers, and gateways, and to interface with a wide variety of standard WANs, LANs, and proprietary networks.

- Chapter 22, “SCC UART Mode,” describes the MPC855T implementation of universal asynchronous receiver transmitter (UART) protocol, used for sending low-speed data between devices.
- Chapter 23, “SCC HDLC Mode,” describes the MPC855T implementation of HDLC protocol.
- Chapter 24, “SCC AppleTalk Mode,” describes the MPC855T implementation of AppleTalk, a set of protocols developed by Apple Computer, Inc. to provide a LAN service between Macintosh computers and printers.
- Chapter 25, “SCC Asynchronous HDLC Mode and IrDA,” describes the asynchronous HDLC and IrDA use of HDLC framing techniques with UART-type characters.
- Chapter 26, “SCC BISYNC Mode,” describes the MPC855T implementation of byte-oriented BISYNC protocol developed by IBM for use in networking products.
- Chapter 27, “SCC Ethernet Mode,” describes the MPC855T implementation of Ethernet protocol.
- Chapter 28, “SCC Transparent Mode,” describes the MPC855T implementation of transparent mode (also called totally transparent mode), which provides a clear channel on which the SCC can send or receive serial data without bit-level manipulation.
- Chapter 29, “Serial Management Controllers (SMCs),” describes two serial management controllers, full-duplex ports that can be configured independently to support one of three protocols—UART, transparent, or general-circuit interface (GCI).
- Chapter 30, “Serial Peripheral Interface (SPI),” describes the serial peripheral interface, which allows the MPC855T to exchange data between other MPC855T chips, the MC68360, the MC68302, the M68HC11 and M68HC05 microcontroller families, and peripheral devices such as EEPROMs, real-time clocks, A/D converters, and ISDN devices.
- Chapter 31, “I2C Controller,” describes the MPC855T implementation of the inter-integrated circuit (I^2C®) controller, which allows data to be exchanged with other I^2C devices, such as microcontrollers, EEPROMs, real-time clock devices, and A/D converters.
- Chapter 32, “Parallel Interface Port (PIP),” describes the parallel interface port which allows data to be sent to and from the MPC855T over 8 or 16 parallel data lines with two handshake control signals.
- Chapter 33, “Parallel I/O Ports,” describes the four general-purpose I/O ports—A, B, C, and D. Each signal in the I/O ports can be configured as a general-purpose I/O signal or as a signal dedicated to supporting communications devices, such as SMCs and SCCs.

- Chapter 34, “CPM Interrupt Controller,” describes how the CPM interrupt controller (CPIC) accepts and prioritizes the internal and external interrupt requests from the CPM blocks and passes them to the system interface unit (SIU). The CPIC also provides a vector during the core interrupt acknowledge cycle.

Suggested Reading

This section lists additional reading that provides background for the information in this manual.

MPC8xx Documentation

Supporting documentation for the MPC855T can be accessed through the world-wide web at http://www.motorola.com. This documentation includes technical specifications, reference materials, and detailed application notes.

Conventions

This document uses the following notational conventions:

Bold	Bold entries in figures and tables showing registers and parameter RAM should be initialized by the user.
mnemonics	Instruction mnemonics are shown in lowercase bold.
italics	Italics indicate variable command parameters, for example, **bcctr***x*. Book titles in text are set in italics.
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote binary number
rA, **r**B	Instruction syntax used to identify a source GPR
rD	Instruction syntax used to identify a destination GPR
REG[FIELD]	Abbreviations or acronyms for registers or buffer descriptors are shown in uppercase text. Specific bits, fields, or numerical ranges appear in brackets. For example, MSR[LE] refers to the little-endian mode enable bit in the machine state register.
x	In certain contexts, such as in a signal encoding or a bit field, indicates a don’t care.
n	Indicates an undefined numerical value
¬	NOT logical operator
&	AND logical operator
|	OR logical operator

Acronyms and Abbreviations

Table i contains acronyms and abbreviations used in this document. Note that the meanings for some acronyms (such as SDR1 and DSISR) are historical, and the words for which an acronym stands may not be intuitively obvious.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
ALU	Arithmetic logic unit
ATM	Asynchronous transfer mode
BD	Buffer descriptor
BIST	Built-in self test
CEPT	Conference des administrations Europeanes des Postes et Telecommunications (European Conference of Postal and Telecommunications Administrations).
C/I	Condition/indication channel used in the GCI protocol
CP	Communications processor
CPM	Communications processor module
DMA	Direct memory access
DPLL	Digital phase-locked loop
DRAM	Dynamic random access memory
DSISR	Register used for determining the source of a DSI exception
EA	Effective address
EEST	Enhanced Ethernet serial transceiver
EPROM	Erasable programmable read-only memory
GCI	General circuit interface
GPCM	General-purpose chip-select machine
GUI	Graphical user interface
HDLC	High-level data link control
I^2C	Inter-integrated circuit
IDL	Inter-chip digital link
IEEE	Institute of Electrical and Electronics Engineers
IrDA	Infrared Data Association
ISDN	Integrated services digital network
JTAG	Joint Test Action Group
LIFO	Last-in-first-out
LRU	Least recently used
LSB	Least-significant byte

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
lsb	Least-significant bit
MAC	Multiply accumulate
MSB	Most-significant byte
msb	Most-significant bit
MSR	Machine state register
NaN	Not a number
NMSI	Nonmultiplexed serial interface
OSI	Open systems interconnection
PCI	Peripheral component interconnect
PPM	Pulse-position modulation
RTOS	Real-time operating system
Rx	Receive
SCC	Serial communications controller
SCP	Serial control port
SDLC	Synchronous Data Link Control
SDMA	Serial DMA
SI	Serial interface
SIU	System interface unit
SMC	Serial management controller
SNA	Systems network architecture
SPI	Serial peripheral interface
SRAM	Static random access memory
TDM	Time-division multiplexed
TE	Terminal endpoint of an ISDN connection
TLB	Translation lookaside buffer
TSA	Time-slot assigner
Tx	Transmit
UART	Universal asynchronous receiver/transmitter
UPM	User-programmable machine
USART	Universal synchronous/asynchronous receiver/transmitter

Chapter 17
Communications Processor Module and CPM Timers

The communications processor module (CPM) provides a flexible and integrated approach to communications-intensive environments. To reduce system frequency and save power, the CPM has its own independent RISC communications processor (CP) that is optimized for serial communications. The CP services several integrated communications channels, performing low-level protocol processing and controlling DMA.

The CPM supports multiple communications channels and protocols, and it has flexible firmware programmability. The CPM frees the core of many computational tasks 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 OSI layer-2 processing, which provides more core bandwidth for higher layer processing.
- By supporting multibuffer memory data structures that are convenient for software handling.

The MPC855T CPM is similar to the one in the MPC860 and both are derived from the CPM in the MC68360 QUICC; see the *MC68360 Quad Integrated Communications Controller (QUICC) User's Manual*.

17.1 Features

Figure 17-1 shows a block diagram of the CPM.

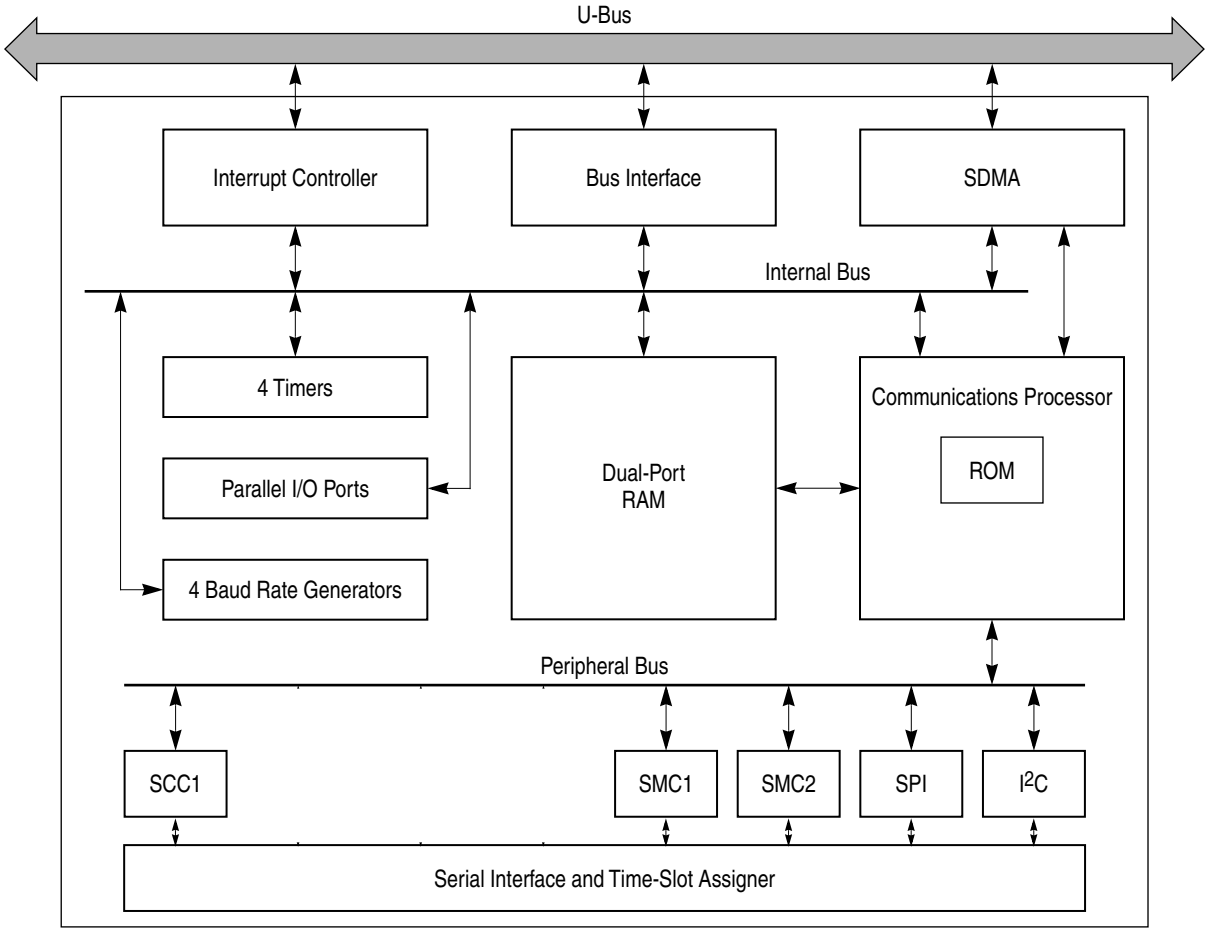


Figure 17-1. CPM Block Diagram

The following lists the CPM's main features:

- Communications processor (CP)
  - Dual-port RAM
  - Internal ROM
  - DMA control for all communications channels
  - Two independent DMA channels for memory-to-memory transfers or interfacing external peripherals
  - RISC timer tables
- A full-duplex serial communications controller (SCC1) that supports the following:
  - UART protocol (asynchronous or synchronous)
  - HDLC protocol
  - AppleTalk protocol
  - Asynchronous HDLC protocol

  - BISYNC protocol
  - Transparent protocol
  - Infrared protocol (IrDA)
  - IEEE802.3/Ethernet protocol
- Two full-duplex serial management controllers (SMCs)
  - UART protocol
  - Transparent protocol
  - GCI protocol for monitor and C/I channels
- Serial peripheral interface (SPI) support for master or slave modes
- Inter-integrated circuit (I^2C) bus controller
- A serial interface (SI) with a time-slot assigner (TSA) that supports multiplexing of data from SCCs and SMCs onto time-division multiplexed (TDM) interface
- Four independent baud rate generators (BRGs)
- Four general-purpose 16-bit timers or two 32-bit timers
- CPM interrupt controller (CPIC)
- General-purpose I/O ports

Figure 17-2 shows a possible MPC855T configuration for a multiprotocol application that supports various communications links and protocols.

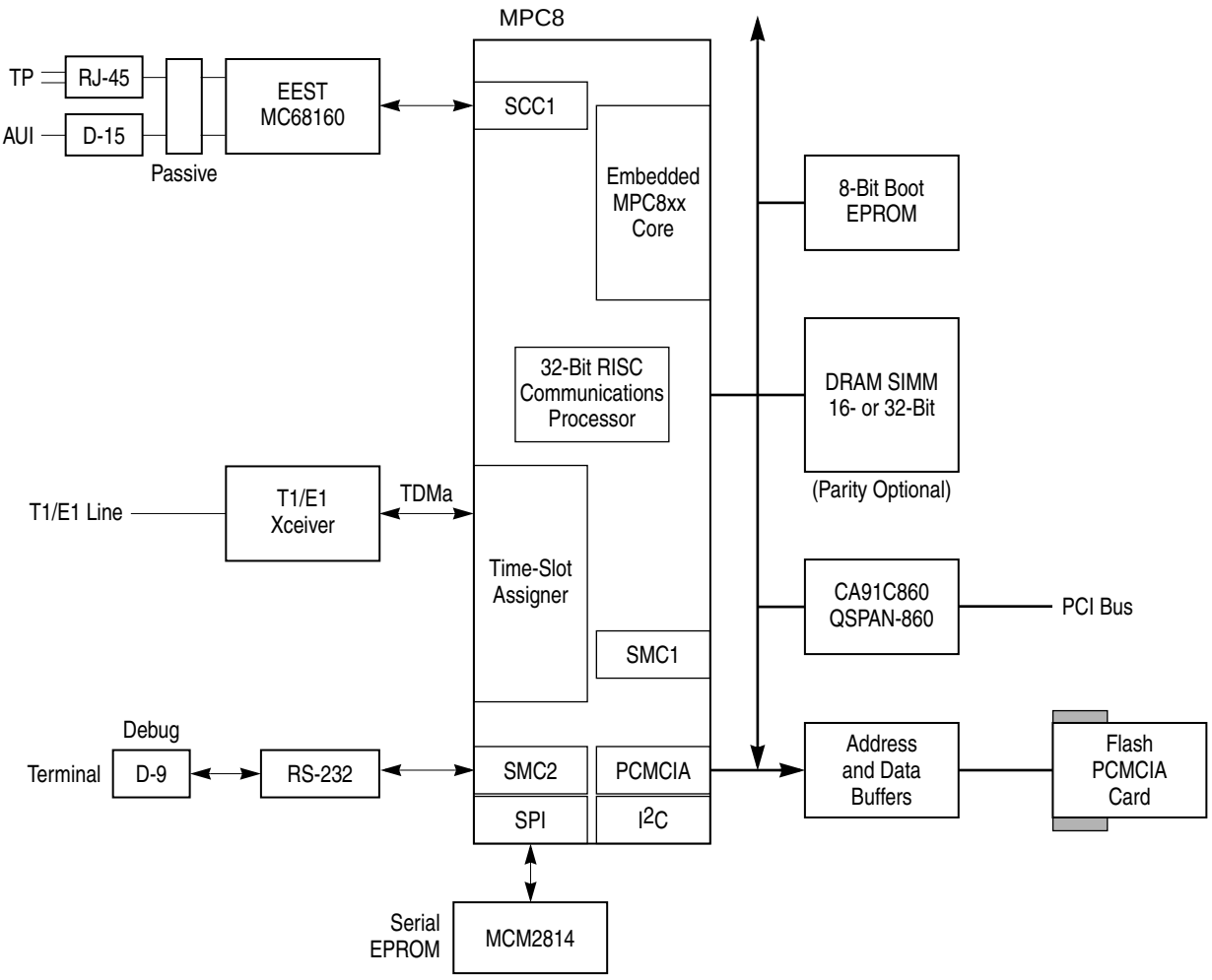


Figure 17-2. MPC855TApplication Design Example

17.2 CPM General-Purpose Timers

The CPM has four identical 16-bit general-purpose timers that can be cascaded into two 32-bit timers. Note that the CPM general-purpose timers are separate and distinct from the RISC timer tables described in Section 18.7, “The RISC Timer Table.” Each timer consists of the following:

- Timer mode register (TMR)
- Timer capture register (TCR)
- Timer counter (TCN)
- Timer reference register (TRR)
- Timer event register (TER)
- Timer global configuration register (TGCR).

Figure is a block diagram of the CPM timers.

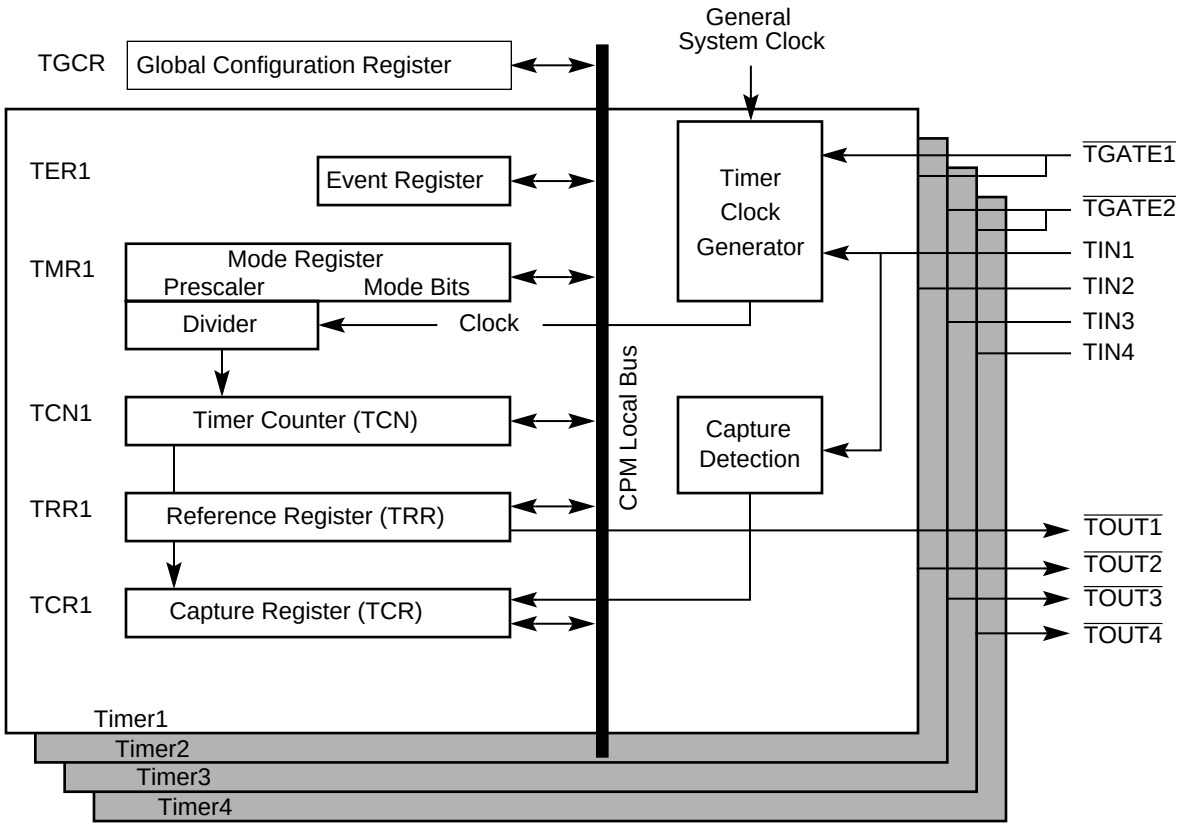


Figure 17-3. CPM Timer Block Diagram

17.2.1 Features

The following list summarizes the main features of the CPM timers:

- Maximum period of 10.7 seconds (at 25 MHz)
- 40-ns resolution (at 25 MHz)
- Programmable sources for the clock input
- Input capture capability
- Output compare with programmable mode for the output pin
- Timers are cascadeable to form 32-bit timers
- Free run and restart modes
- Functionally compatible with MC68360 timers
- Timer 1 is used with the PCMCIA speaker input to generate alerts on SPKROUT.

17.2.2 CPM Timer Operation

The following subsections describe the timer operation. The timer mode registers (TMR*x*) and the timer global configuration register (TGCR) mentioned in this section are described in Section 17.2.3, “CPM Timer Register Set.”

17.2.2.1 Timer Clock Source

The clock input to the prescaler can be selected from three sources:

- The general system clock
- The general system clock divided by 16
- An external source on the corresponding TIN*x* pin

The general system clock (GCLK2) is generated in the clock synthesizer. To save power, the general system clock can be divided before it leaves the clock synthesizer (slow-go mode). Regardless of the resulting general system clock frequency, either that frequency or that frequency divided by 16 can be chosen as the input to the prescaler of each timer. Also, an external clock source can be supplied on the TIN*x* signal. If two 16-bit timers are cascaded internally into a 32-bit timer, one timer uses the clock generated by the output of another timer.

The clock input source is selected by TMR*x*[ICLK]. The prescaler is programmed in TMR*x*[PS] and divides the clock input by values between 1 and 256; the prescaler output is used as an input to the 16-bit counter. The best resolution of the timer is one clock cycle (40 ns at 25 MHz). The maximum period is 268,435,456 cycles, which is 10.7 seconds at 25 MHz.

17.2.2.2 Timer Reference Count

TMR*x*[FRR] (the free-run/restart bit) can be configured so that when a reference is reached the count either continues or begins again. When the reference value is reached, the corresponding TER*x* event bit is set and an interrupt is issued if TMR*x*[ORI] = 1. Also when the reference value is reached, the timers can output a signal on their timer output pins ($\overline{\text{TOUT}}$[1–4]). The output signal can be programmed to be an active-low pulse or a toggle of the current output as selected by TMR*x*[OM] (the output mode bit).

17.2.2.3 Timer Capture

Each timer’s 16-bit capture register, TCR*x*, is used to latch the value of the counter when a defined transition of TIN*x* is sensed by the corresponding input capture edge detector. The type of transition triggering the capture is selected by TMR*x*[CE]. When a capture or reference event occurs, the corresponding TER*x* event bit is set and a maskable interrupt request is issued to the CPIC.

17.2.2.4 Timer Gating

Timers can be gated or restarted by one of two external gate signals—$\overline{\text{TGATE1}}$for timer 1 and/or 2, $\overline{\text{TGATE2}}$ for timer 3 and/or 4. Normal gate mode enables the count on a falling edge of $\overline{\text{TGATE}x}$ and disables the count on the rising edge of $\overline{\text{TGATE}x}$. This allows the timer to count conditionally, depending on the state of $\overline{\text{TGATE}x}$.

Restart gate mode is like normal gate mode, but also resets the counter on the falling edge of $\overline{\text{TGATE}x}$. The restart gate mode can be used for pulse interval measurement and bus monitoring:

- Pulse measurement—The restart gate mode can measure a low pulse on $\overline{\text{TGATE}x}$. The rising edge of $\overline{\text{TGATE}x}$ completes the measurement. If $\overline{\text{TGATE}x}$ is externally connected to TIN*x*, 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 $\overline{\text{TGATE}x}$. 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 TMR; the gate operating mode is selected in the TGCR.

Note that $\overline{\text{TGATE}x}$ is internally synchronized to the system clock. However, if $\overline{\text{TGATE}x}$ meets the asynchronous input setup time, the counter begins counting after one system clock when the input clock source (TMR*x*[ICLK]) is internal.

17.2.2.5 Cascaded Mode

Timer 1 can be internally cascaded to timer 2 and timer 3 can be internally cascaded to timer 4 to form 32-bit timers. The TGCR is used to put the timers into cascaded mode, as shown in Figure 17-4.

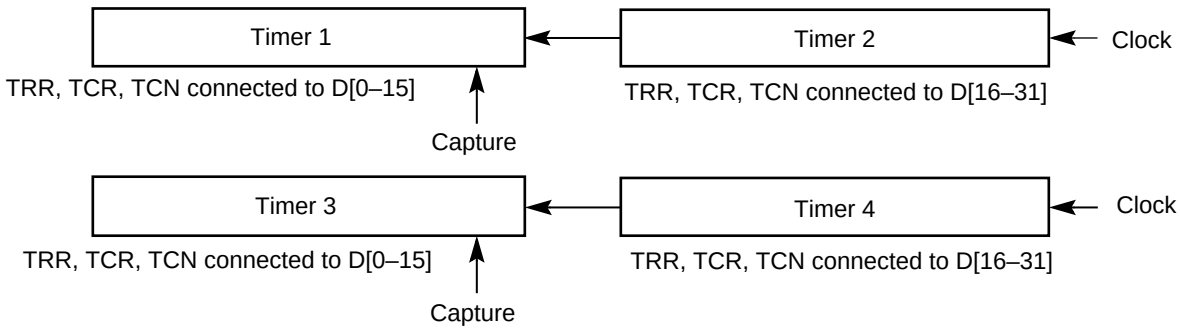


Figure 17-4. Timer Cascaded Mode Block Diagram

If TGCR[CAS*x*] is set, the two corresponding timers function as a 32-bit timer with a 32-bit TRR, TCR, and TCN. In this case, the mode registers TMR1 and TMR3 are ignored and TMR2 and TMR4 are used to define the mode. Similarly, the capture is controlled by TIN2

or TIN4, and interrupts are generated by TER2 or TER4. In cascaded mode, the cascaded TRR, TCR, and TCN should always be accessed with 32-bit bus cycles.

17.2.2.6 Timer 1 and SPKROUT

Timer 1 can be used to drive audio alerts through the PCMCIA SPKROUT signal. Enabling timer 1 results in SPKROUT being driven with timer 1's frequency. Timer 1 is exclusive ORed with the 4 $\overline{\text{SPKR\_B}}$ input signals to generate SPKROUT.

To prevent timer 1 from affecting SPKROUT, either use the timer in a pulse mode or do not enable it.

17.2.3 CPM Timer Register Set

The following subsections discuss the CPM timer register set.

17.2.3.1 Timer Global Configuration Register (TGCR)

The timer global configuration register (TGCR) contains configuration parameters used by all four timers. It allows simultaneous starting and stopping of any number of timers as long as one bus cycle is used to access TGCR.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	**CAS4**	**FRZ4**	**STP4**	**RST4**	**GM2**	**FRZ3**	**STP3**	**RST3**	**CAS2**	**FRZ2**	**STP2**	**RST2**	**GM1**	**FRZ1**	**STP1**	**RST1**
Reset	0															
R/W	R/W															
Addr	0x980															

Figure 17-5. Timer Global Configuration Register (TGCR)

This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Table 17-1 describes the TGCR fields.

Table 17-1. TGCR Field Descriptions

Bits	Name	Description
0	**CAS4**	Cascade timers. 0 Normal operation. 1 Timers 3 and 4 are cascaded to form a 32-bit timer.
1, 5, 9, 13	**FRZ*x***	Freeze timer *x*. 0 The corresponding timer ignores the FRZ state. 1 Stops the corresponding timer if the MPC855T enters FRZ state. FRZ state is entered in debug mode as defined in Chapter 44, “System Development and Debugging.”
2, 6, 10, 14	**STP*x***	Stop timer *x*. 0 Normal operation. 1 Stop the timer. This bit stops all clocks to the timer, except the U-bus interface clock allowing the timer registers to be read or written. The clocks to the timer remain inactive until this bit is cleared or a hardware reset occurs.

Table 17-1. TGCR Field Descriptions (continued)

Bits	Name	Description
3, 7, 11, 15	**RST*x***	Reset timer *x*. The associated TMR*x* and TRR*x* registers should be initialized before enabling the timer with RST*x*. 0 Reset the corresponding timer. Upon clearing this bit, all associated timer registers are reset. 1 Enable the corresponding timer if STP is cleared.
4	**GM2**	Gate mode for $\overline{\text{TGATE2}}$. Valid only if TMR3[GE] or TMR4[GE] is set. 0 Restart gate mode. A falling edge of $\overline{\text{TGATE2}}$ enables and restarts the count and a rising edge of $\overline{\text{TGATE2}}$ disables the count. 1 Normal gate mode. This mode is the same as 0, except the falling edge of $\overline{\text{TGATE2}}$ does not restart the count value in the TCN.
8	**CAS2**	Cascade timers. 0 Normal operation. 1 Timers 1 and 2 are cascaded to form a 32-bit timer.
12	**GM1**	Gate mode for $\overline{\text{TGATE1}}$. Valid only if TMR1[GE] or TMR2[GE] is set. 0 Restart gate mode. A falling $\overline{\text{TGATE1}}$ 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 $\overline{\text{TGATE1}}$ does not restart the count value in the TCN.

17.2.4 Timer Mode Registers (TMR1–TMR4)

The timer mode registers (TMR1–TMR4), shown in Figure 17-6, are identical. Before modifying TMR*x*, reset the timer by clearing TGCR[RST*x*].

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															
R/W	R/W															
Addr	0x990 (TMR1), 0x992 (TMR2), 0x9A0 (TMR3), 0x9A2 (TMR4)															

Figure 17-6. Timer Mode Registers (TMR1–TMR4)

These registers are affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Table 17-2 describes the TMR fields.

Table 17-2. TMR1–TMR4 Field Descriptions

Bits	Name	Description
0–7	**PS**	Prescaler value. The prescaler is programmed to divide the clock input by a value between 1 and 256. A 0x00 value divides the clock by 1; 0xFF divides it by 256.
8–9	**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.

Table 17-2. TMR1–TMR4 Field Descriptions (continued)

Bits	Name	Description
10	**OM**	Output mode. 0 Active-low pulse on $\overline{\text{TOUTx}}$ for one timer input clock cycle as defined by ICLK. Thus, $\overline{\text{TOUTx}}$ may be low for one general system clock period, one general system clock/16 period, or one TINx clock cycle period. Changes to $\overline{\text{TOUTx}}$ occur on the falling edge of the system clock. 1 Toggle $\overline{\text{TOUTx}}$. Changes to $\overline{\text{TOUTx}}$ occur on the falling edge of the system clock.
11	**ORI**	Output reference interrupt enable. 0 Disable interrupt for reference that is reached. Does not affect an interrupt on the capture function. 1 Enable interrupt when the reference value is reached.
12	**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.
13–14	**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 TIN*x* signal (falling edge).
15	**GE**	Gate enable. 0 $\overline{\text{TGATEx}}$ is ignored. 1 $\overline{\text{TGATEx}}$ is used to control the timer—$\overline{\text{TGATE1}}$ for timer 1 and 2, $\overline{\text{TGATE2}}$ for timer 3 and 4.

17.2.4.1 Timer Reference Registers (TRR1–TRR4)

Each timer reference register (TRR1–TRR4), shown in Figure 17-7, contains the timeout's reference value. The reference value is not reached until TCN*x* increments to equal the timeout reference value. These registers are affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	**Timeout reference value**															
Reset	0xFFFF															
R/W	R/W															
Addr	0x994 (TRR1), 0x996 (TRR2), 0x9A4 (TRR3), 0x9A6 (TRR4)															

Figure 17-7. Timer Reference Registers (TRR1–TRR4)

17.2.4.2 Timer Capture Registers (TCR1–TCR4)

Each timer capture register (TCR1–TCR4), shown in Figure 17-8, is used to latch the value of the counter according to TMR*x*[CE].These registers are affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	Latched counter value															
Reset	0															
R/W	R/W															
Addr	0x998 (TCR1), 0x99A (TCR2), 0x9A8 (TCR3), 0x9AA (TCR4)															

Figure 17-8. Timer Capture Registers (TCR1–TCR4)

17.2.4.3 Timer Counter Registers (TCN1–TCN4)

Each timer counter register (TCN1–TCN4), shown in Figure 17-9, is an up-counter. A read cycle to TCN1–TCN4 yields the current value of the timer, but does not affect the counting operation. A write cycle to TCN1–TCN4 sets the register to the written value, thus causing its corresponding prescaler, TMR*x*[PS], to be reset.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	**Up counter**															
Reset	0															
R/W	R/W															
Addr	0x99C (TCN1), 0x99E (TCN2), 0x9AC (TCN3), 0x9AE (TCN4)															

Figure 17-9. Timer Counter Registers (TCN1–TCN4)

Note that the counter registers may not be updated correctly if a write is made while the timer is not running. Use TRR*x* to define the preferred count value.These registers are affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

17.2.4.4 Timer Event Registers (TER1–TER4)

Each timer event register (TER1–TER4), shown in Figure 17-10, reports events recognized by the timers. When an output reference event is recognized, the timer sets TER*x*[REF] regardless of the corresponding TMR*x*[ORI]. The capture event is set only if it is enabled in TMR*x*[CE].

Writing ones clears event bits; writing zeros has no effect. Both event bits must be cleared before the timer negates the interrupt to the CPIC. These registers are affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—														REF	CAP
Reset	0															
R/W	R/W															
Addr	0x9B0 (TER1), 0x9B2 (TER2), 0x9B4 (TER3), 0x9B6 (TER4)															

Figure 17-10. Timer Event Registers (TER1–TER4)

Table 17-3 describes the TER fields.

Table 17-3. TER Field Descriptions

Bits	Name	Description
0–13	—	Reserved, should be cleared.
14	REF	Output reference event. When set, indicates the counter reached the value in the TRR. TMR[ORI] is used to enable the interrupt request caused by this event.
15	CAP	Capture event. Indicates that the counter value has been latched into the TCR. TMR[CE] enables generation of this event.

17.2.5 Timer Initialization Examples

The following two initialization sequences program timer 2 to generate an interrupt every 10 µs. The first sequence uses timer 2 alone, while the second example uses timers 1 and 2 in cascaded mode. Assuming a 25-MHz general system clock, an interrupt should be generated every 250 system clocks.

1. Set TGCR = 0x0000 to reset timer 2.
2. Set TMR2 = 0x001A to set the prescaler to divide by 1 and the clock source to the general system clock. This value also enables an interrupt when the timer reaches the reference count and immediately clears (restarts) the TCN for the next interrupt.
3. Set TCN2 = 0x0000 to clear the timer 2 counter (default).
4. Set TRR2 = 0x00FA to initialize the timer 2 reference value to 250.
5. Write TER2 = 0xFFFF to clear TER2 of any previous events.
6. Set CIMR = 0x0004_0000 to enable timer 2 interrupts in the CPIC and initialize the CICR.
7. Set TGCR = 0x0010 to enable timer 2 to begin counting.

To implement the same function with a 32-bit timer using timers 1 and 2, follow these steps:

1. Set TGCR = 0x0080. Cascade timers 1 and 2 and put them in reset state.
2. Set TMR2 = 0x001A to set the prescaler to divide by 1 and the clock source to the general system clock. This value also enables an interrupt when the timer reaches the reference count and immediately clears the TCN for the next interrupt.
3. Set TMR1 = 0x0000. Enable timer 1 to use the timer 2 output as its input (TMR1[ICLK] = 0b00).
4. Set TCN1 = 0x0000 and TCN2 = 0x0000. Initialize the count of the combined timers 1 and 2 to zero (TMR1 default) by using one 32-bit data move to TCN1.
5. Set TRR1 = 0x0000 and TRR2 = 0x00FA. Initialize the reference value of the combined timers 1 and 2 to 250 by using one 32-bit data move to TRR1.
6. Write TER2 = 0xFFFF to clear TER2 of any previous events.

7. Set CIMR = 0x0004_0000 to enable timer 2 interrupts in the CPIC and initialize the CICR.
8. Set TGCR = 0x0091 to enable timers 1 and 2 to begin counting in cascaded mode.

Chapter 18
Communications Processor

Transacting with the communications peripherals on a separate bus from the MPC8xx core, the CPM's 32-bit communications processor (CP) handles the low-level communications tasks, freeing the core for higher-level tasks. The CP implements the chosen protocols using the serial controllers and parallel interface port and manages the data transfer through the serial DMA (SDMA) channels between the I/O channels and memory. It also manages IDMA (independent DMA) channels and contains an internal timer used to implement additional software timers.

The CP's architecture and instruction set are optimized for data communications and processing required by many wire-line and wireless communications standards.

18.1 Features

The following lists the CP's main features:

- Performs lower-layer protocol processing for communication channels
- Protocol-processing microcode routines located in internal ROM
- Optional Motorola-supplied microcode packages run from dual-port RAM (The microcode packages allow the addition of protocols and other enhancements.)
- Supports general-purpose DMA using two IDMA channels
- Supports DMA bursting for memory-to-memory IDMA
- Performs DMA of serial data to external memory
- RISC timer table supports a maximum of 16 software timers

Figure 18-1 is a block diagram of the CP.

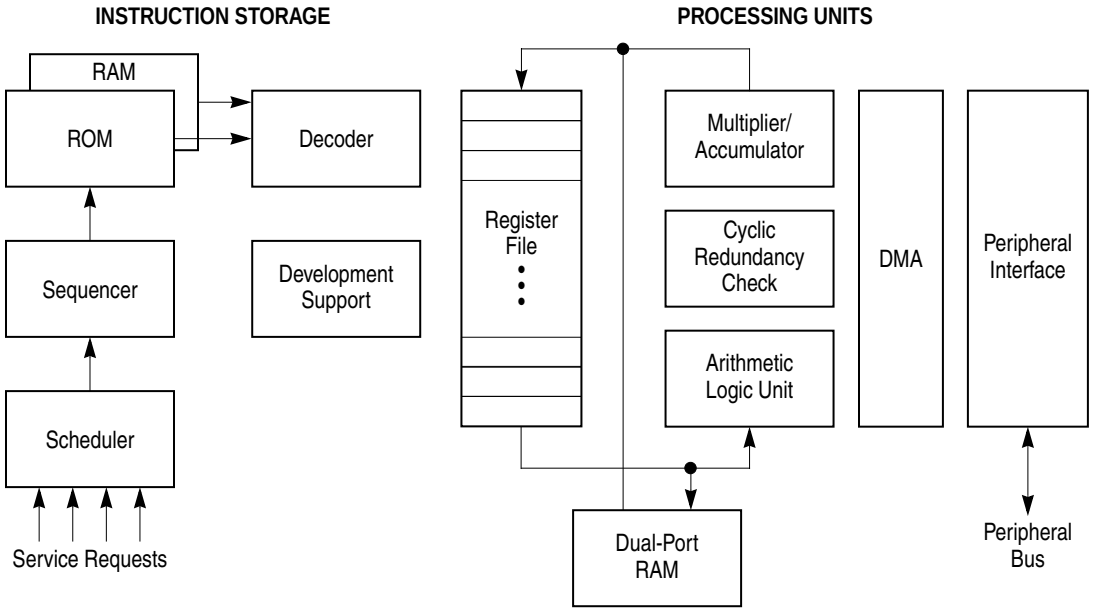


Figure 18-1. Communications Processor (CP) Block Diagram

18.2 Communicating with the Core

The CP communicates with the core in the following ways:

- By exchanging parameters using the dual-port RAM.
- By executing special commands that are issued by the core through the CP command register (CPCR).
- By generating interrupts using the CPM interrupt controller (CPIC).
- By letting the core read the CPM status/event registers at any time.

18.3 Communicating with the Peripherals

The CP uses the peripheral bus to communicate with the peripherals. The serial communications controller (SCC1) has a separate receive and transmit FIFO. The SCC1 receive and transmit FIFOs are 32 bytes each; . The serial management controllers (SMCs), serial peripheral interface (SPI), and I^2C are all double-buffered, creating effective FIFO sizes of two characters.The parallel interface port (PIP) is a single register interface.

Table 18-1 shows the order in which the CP handles requests from peripherals from highest to lowest priority.

Table 18-1. Peripheral Prioritization

Priority	Request
1	Reset in the CPCR or $\overline{\text{SRESET}}$
2	SDMA bus error

Table 18-1. Peripheral Prioritization (continued)

Priority	Request
3	Commands issued to the CPCR
4	IDMA emulation: $\overline{\text{DREQ0}}$ (default—option 1) [1]
5	IDMA emulation: $\overline{\text{DREQ1}}$ (default—option 1)[1]
6	SCC1 Rx
7	SCC1 Tx
8	IDMA emulation: $\overline{\text{DREQ0}}$ (option 2)[1]
9	IDMA emulation: $\overline{\text{DREQ1}}$ (option 2)[1]
10	SMC1 Rx
11	SMC1 Tx
12	SMC2 Rx
13	SMC2 Tx
14	SPI Rx
15	SPI Tx
16	I^2C Rx
17	I^2C Tx
18	PIP
19	RISC timer table
20	IDMA emulation: $\overline{\text{DREQ0}}$ (option 3)[1]
21	IDMA emulation: $\overline{\text{DREQ1}}$ (option 3)[1]

[1] See the RCCR[DRQP] description in Section 18.5.1, “RISC Controller Configuration Register (RCCR).”

18.4 CP Microcode Revision Number

In addition to the microcode routines located in internal ROM, the CP can execute protocol-enhancing microcode packages from dual-port RAM. The CP writes a part revision number stored in ROM to a dual-port RAM location called REV_NUM that resides in the miscellaneous parameter RAM. REV_NUM determines which version of Motorola-supplied microcode package to use; see Table 18-2.

Table 18-2. CP Microcode Revision Number

Offset [1]	Name	Width	Description
0x00	REV_NUM	Half-word	Microcode revision number
0x02	—	Half-word	Reserved

Table 18-2. CP Microcode Revision Number (continued)

0x04	—	Word	Reserved
0x08	—	Word	Reserved

1 Offset from the base of the miscellaneous parameter area (at offset 0x1CB0 of the dual-port RAM).

For the latest documentation on part/revision numbers and microcode REV_NUMs, see the website at http://www.motorola.com.

18.5 CP Register Set and CP Commands

The following sections describe the communications processor registers and commands.

18.5.1 RISC Controller Configuration Register (RCCR)

The RISC controller configuration register (RCCR), shown in Figure 18-2, tells the CP to run microcode from ROM or dual-port RAM and controls the CP's internal timer. It also sets the IDMA request modes and priority.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	TIME	—	TIMEP						DR1M	DR0M	DRQP		EIE	SCD	ERAM	
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	0x9C4															

Figure 18-2. RISC Controller Configuration Register (RCCR)

This register is affected by $\overline{\text{HRESET}}$ but is not affected by $\overline{\text{SRESET}}$. Table 18-3 describes the RCCR fields.

Table 18-3. RCCR Field Descriptions

Bits	Name	Description
0	TIME	Timer enable. Controls whether the CP's internal timer sends a tick to the CP based on the value programmed in the timer period (TIMEP). 0 Stop RISC timer table scanning. 1 Start RISC timer table scanning.
1	—	Reserved. Should be cleared.
2–7	TIMEP	Timer period. Controls the period of the CP's internal timer tick. The RISC timer table are scanned on each timer tick. The input to the timer tick generator is the system clock divided by 1,024. The formula is: timer tick period = (TIMEP + 1) × 1,024 system clocks. Thus, a value of 0 stored in this field creates a timer tick every 1 × (1,024) = 1,024 system clocks; a value of 63 causes a tick every 64 × (1,024) = 65,536 system clocks.
8	DR1M	IDMA request 1 mode. Controls the IDMA request 1 ($\overline{\text{DREQ1}}$) sensitivity mode. See Section 19.3.7, "IDMA Interface Signals—DREQ and SDACK." 0 $\overline{\text{DREQ1}}$ is edge-sensitive. 1 $\overline{\text{DREQ1}}$ is level-sensitive.

Table 18-3. RCCR Field Descriptions (continued)

Bits	Name	Description
9	DR0M	IDMA request 0 mode. Controls the IDMA request 0 ($\overline{\text{DREQ0}}$) sensitivity mode. See Section 19.3.7, “IDMA Interface Signals—DREQ and SDACK.” 0 $\overline{\text{DREQ0}}$ is edge-sensitive. 1 $\overline{\text{DREQ0}}$ is level-sensitive.
10–11	DRQP	IDMA emulation request priority. Controls the priority of the external request signals that relate to the serial channels. See Section 18.3, “Communicating with the Peripherals.” 00 IDMA requests have priority over the SCC. 01 IDMA requests have priority immediately following the SCC (option 2). 10 IDMA requests have the lowest priority (option 3). 11 Reserved.
12	EIE	External interrupt enable. Configure as instructed in the download process of a Motorola-supplied RAM microcode package. 0 $\overline{\text{DREQ0}}$ cannot interrupt the CP. 1 $\overline{\text{DREQ0}}$ will interrupt the CP.
13	SCD	Scheduler configuration. Configure as instructed in the download process of a Motorola-supplied RAM microcode package. 0 Normal operation. 1 Alternate configuration of the scheduler.
14–15	ERAM	Enable RAM microcode Configure as instructed in the download process of a Motorola-supplied microcode package. See Section 18.6.1, “System RAM and Microcode Packages.” 00 Disable microcode program execution in the dual-port system RAM. 01 Microcode executes from the first 512 bytes and a 256-byte extension of dual-port system RAM. 10 Microcode executes from the first 1 Kbyte and a 256-byte extension of dual-port system RAM. 11 Microcode executes from the first 2 Kbytes and a 512-byte extension of dual-port system RAM.

18.5.2 RISC Microcode Development Support Control Register (RMDS)

The RISC microcode development support control register (RMDS), shown in Figure 18-3, determines which regions of the dual-port RAM can contain executable microcode. RMDS is used with RCCR[ERAM] to determine the valid address space for executable microcode. Section 18.6.1, “System RAM and Microcode Packages,” describes the partitioning of the dual-port system RAM.

Bit	0	1	2	3	4	5	6	7
Field	ERAM4K	—						
Reset	0000_0000_0000_0000							
R/W	R/W							
Addr	0x9C7							

Figure 18-3. RISC Microcode Development Support Control Register (RMDS)

RMDS fields are described in Table 18-4.

Table 18-4. RMDS Field Descriptions

Bits	Name	Description
0	ERAM4K	Enable RAM microcode (at offset 4K) 0 Microcode may be executed only from the first 2 Kbytes of the dual-port RAM. 1 Microcode is also executed from the 2 Kbytes of the second half of the dual-port RAM with a 512-byte extension.
1–7	—	Reserved, should be cleared.

18.5.3 CP Command Register (CPCR)

The MPC8xx core can issue commands to control communications via the CP command register (CPCR), shown in Figure 18-4. The CP commands handle special cases, such as initializing or stopping a channel, and are protocol-dependent.

When the core issues a command it sets CPCR[FLG]. When the command completes, the CP clears FLG to signal the core for the next command. The core must wait for FLG to be cleared before issuing another CP command. The core can, however, issue the CP reset command (CPCR = 0x8001) at any time, regardless of FLG.

Note that the CPCR has a different bit format for ATM operations; see Section 38.3, “ATM Commands.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	RST	—			OPCODE				CH_NUM				—			FLG
Reset	0															
R/W	R/W															
Addr	0x9C0															

Figure 18-4. CP Command Register (CPCR)

This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Table 18-5 describes CPCR fields.

Table 18-5. CPCR Field Descriptions

Bits	Name	Description
0	RST	CP reset command. Set by the core and cleared by the CP. Executing this command clears RST and FLG within two general system clocks. The CPM reset routine takes approximately 60 clocks, but CPM initialization can start immediately after this command is issued. Use RST to reset the registers and parameters for all the channels (SCC, SMCs, SPI, I^2C) as well as the CPM and RISC timer table. RST does not, however, affect the serial interface or parallel I/O registers. 0 No reset issued. 1 Reset issued.
1–3	—	Reserved. Should be cleared.
4–7	OPCODE	Operation code for the core-issued CP commands. See Table 18-6.

Table 18-5. CPCR Field Descriptions (continued)

Bits	Name	Description
8–11	CH_NUM	Channel number. Defines the specific sub-block on which the command is to operate. Some sub-blocks share channel number encodings if their commands are mutually exclusive. 0000 SCC1 0001 I^2C/IDMA1 001x Reserved 0100 Reserved 0101 SPI/IDMA2/RISC timers 011x Reserved 1000 Reserved 1001 SMC1 101x Reserved 1100 Reserved 1101 SMC2 111x Reserved
12–14		Reserved. Should be cleared.
15	FLG	Command semaphore flag. Set by the core and cleared by the CP. 0 CP is ready for a new command. 1 CP is currently processing a command—cleared when the command is done or after reset.

18.5.4 CP Commands

A given CP command opcode may have different meanings, or may not apply, depending on which channel (sub-block) the command targets.

Table 18-6 describes the CP commands.

Table 18-6. CP Commands

Command	Description
INIT TX AND RX PARAMS	Initialize transmit and receive parameters. Initializes the CP's temporary Tx and Rx parameters in the parameter RAM to the user-defined reset values—often required when switching protocols.
INIT RX PARAMS	Initialize receive parameters. Initializes the CP's temporary Rx parameters in the parameter RAM to the user-defined reset values—often required when switching protocols.
INIT TX PARAMS	Initialize transmit parameters. Initializes the CP's temporary Tx parameters in the parameter RAM to the user-defined reset values—often required when switching protocols.
ENTER HUNT MODE	Causes the receiver to stop and wait for a new frame—exact operation depends on the protocol used.
STOP TX	Stop transmission. Stops the transmitting channel as soon as the Tx FIFO has been emptied. It should only be used when transmission needs to be stopped as quickly as possible. Transmission continues when RESTART TX is issued.
GRACEFUL STOP TX	Graceful stop transmission. Stops the transmitting channel after the whole current frame has been sent. Transmission continues when RESTART TX is issued and the ready bit is set in the next TxBD.
RESTART TX	Restart transmission. After STOP TX or GRACEFUL STOP TX, RESTART TX starts the transmitter, which begins polling the R bit of the current BD.

Table 18-6. CP Commands (continued)

Command	Description
CLOSE RX BD	Closes the current RxBD in mid-reception; reception continues using the next available BD. Use CLOSE RX BD to access the data buffer without waiting for the SCC to finish filling it.
INIT IDMA	Initialize IDMA transfers. Initializes the IDMA internal CP state to the user-defined reset value.
STOP IDMA	Stop IDMA transfers. The CP terminates current IDMA transfers.
SET TIMER	Used to activate, deactivate, or reconfigure the 16 timers of the RISC timer table.
SET GROUP ADDRESS	Sets a hash table bit for the Ethernet logical group address recognition function.
GCI ABORT REQUEST	GCI receiver sends an abort request.
GCI TIMEOUT	Performs the GCI timeout function.
RESET BCS	Used in BISYNC mode to reset the block check sequence calculation.
ATM Commands	See Section 38.3, “ATM Commands.”
U	Undefined. Reserved for use by Motorola-supplied RAM microcode packages.

18.5.4.1 CP Command Examples

To completely reset the CPM, write 0x8001 to the CPCR. After the FLG bit has cleared, set-up of the CPM can continue. To execute ENTER HUNT MODE on SCC1, for example, write 0x0301 to the CPCR. While the command is executing, the CPCR returns a 0x0301 value; after executing, it returns 0x0300.

18.5.4.2 CP Command Execution Latency

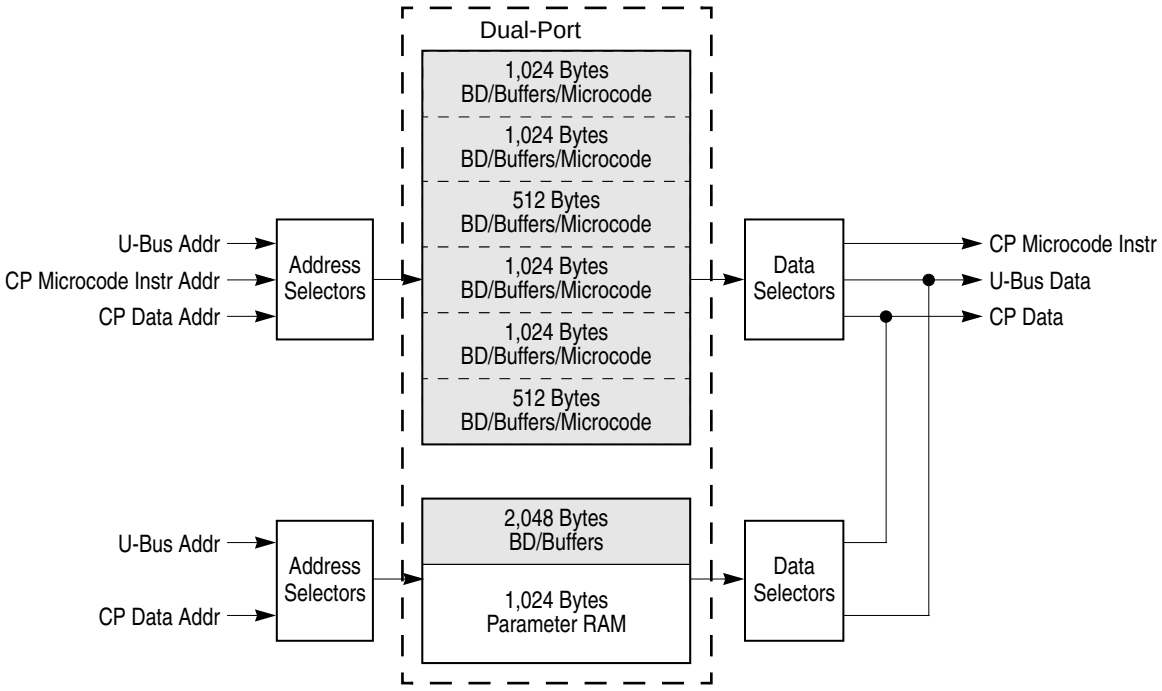
The worst-case command execution latency for the CP is 500 clocks, while the typical command execution latency is about 40 clocks.

18.6 Dual-Port RAM

The CPM has 8 Kbytes of static RAM configured as dual-port memory, shown in Figure 18-5.

NOTE

The entire dual-port RAM should be cleared as the first step in system initialization. This step should be followed by issuing a CPM reset using the CPCR. Only after these two steps should the dual-port RAM be programmed for specific CPM functions.



Shaded area is system RAM. Note that in this figure, the area is not contiguous memory. For an accurate representation of the physical implementation, see Figure 18-6.

Figure 18-5. Dual-Port RAM Block Diagram

The dual-port RAM consists of 7 Kbytes of system RAM (see Section 18.6.1, “System RAM and Microcode Packages”) and 1 Kbyte of parameter RAM (see Section 18.6.3, “Parameter RAM”) and is used for the following:

- Storing parameters associated with the SCC, SMCs, SPI, I²C, and IDMAs (in parameter RAM only)
- Storing the BDs (in any unused dual-port RAM area)
- Storing buffers (in any unused dual-port RAM area or external memory)
- Storing Motorola-supplied microcode for the CP (in system RAM only)
- Scratch pad area for user software (in any unused dual-port RAM area)

The dual-port RAM can be accessed either by the CP or by one of two internal U-bus masters—the MPC8xx core or an SDMA channel. The core and the SDMA channels access the dual-port RAM in two clocks, while the CP takes only one clock. For simultaneous accesses with at least one write operation, the CP is delayed by one clock.

When the core or SDMA channel access the dual-port RAM, the data and address are passed through the U-bus. The CP can fetch data from the entire dual-port RAM and microcode instructions from portions of the system RAM.

The controller and sub-block parameters of the parameter RAM and the optional microcode packages in system RAM use fixed addresses. The buffer descriptors, buffers, and scratch pad area, however, can be located in any unused dual-port RAM area. See Figure 18-6.

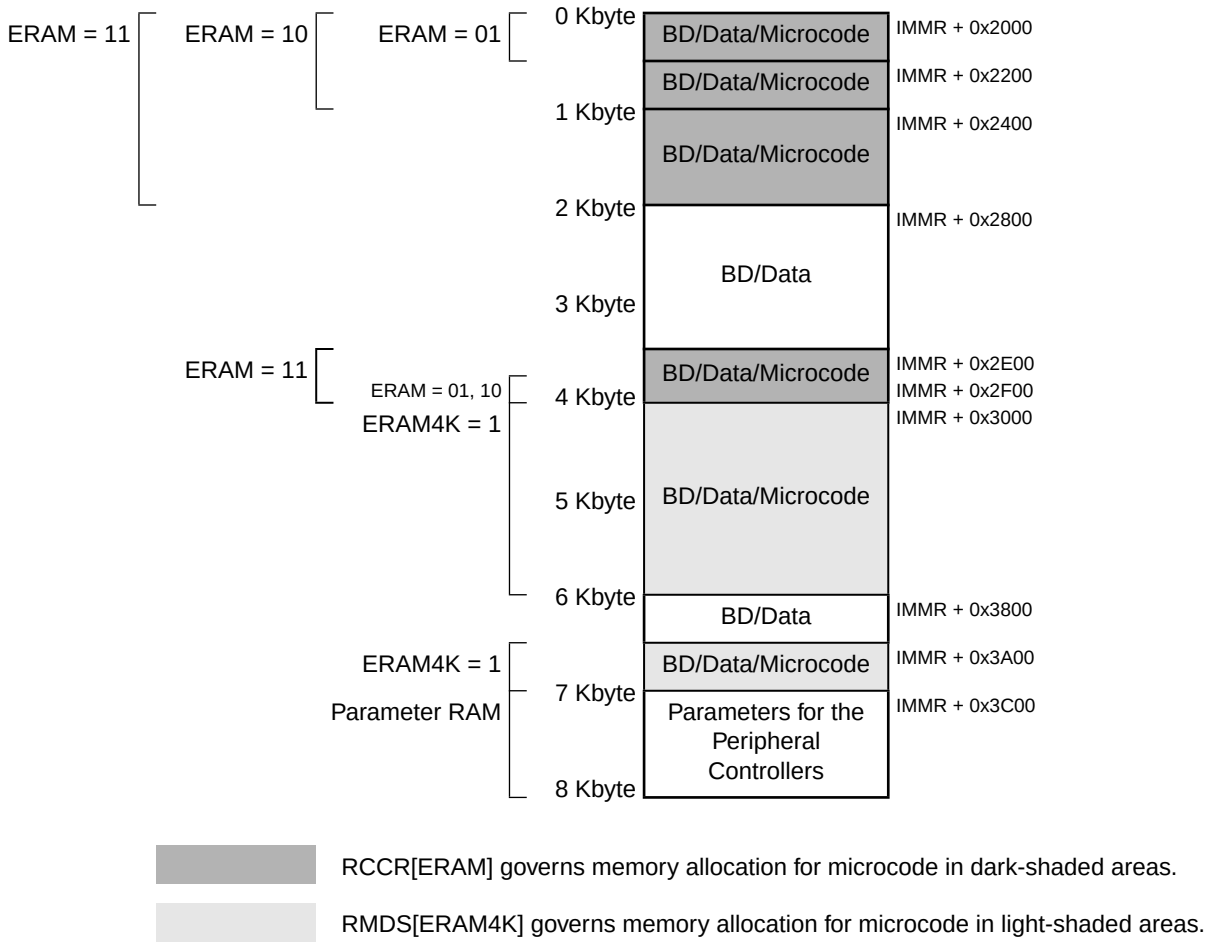


Figure 18-6. Dual-Port RAM Memory Map

18.6.1 System RAM and Microcode Packages

When optional Motorola-supplied RAM microcode packages are activated, certain portions of the 7-Kbyte system RAM are no longer available. (The 1-Kbyte parameter RAM is not affected.) Depending on the memory requirements of the microcode package, some or all of the shaded areas of Figure 18-6 become locked. Reads to locked areas return all ones. The unshaded 2-Kbyte(non-contiguous) area of system RAM is always available to the user.

The enable-RAM-microcode field of the RISC configuration register, RCCR[ERAM], selects the three possible configurations for microcode area sizes—first 512-byte block, first two 512-byte blocks, or first four 512-byte blocks. When just the first and/or second

512-byte blocks are used for microcode, an additional 256-byte extension of system RAM is also locked. When all four 512-byte blocks are used for microcode, an additional 512-byte extension of system RAM is locked. See the darker-shaded areas of Figure 18-6.

In addition to RCCR[ERAM], RMDS[ERAM4K] (enable RAM microcode at offset 4K) affects the system RAM memory configuration for microcode packages. Setting RMDS[ERAM4K] locks a 2-Kbyte block and a 512-byte extension (the lighter-shaded areas of Figure 18-6) for microcode execution.

18.6.2 The Buffer Descriptor (BD)

The SCC, SMCs, SPI, IDMA, PIP, and I²C use buffer descriptors (BDs) to define the interface to buffers. BDs can be placed in any unused area of the dual-port RAM. Table 18-7 shows the general BD structure common to these controllers.

Table 18-7. General BD Structure

BD Base Offset	Field
0x00	Status and control
0x02	Data length
0x04	High-order of buffer pointer
0x06	Low-order of buffer pointer

18.6.3 Parameter RAM

The CPM maintains a section of dual-port RAM called the parameter RAM. It contains parameters for SCC, SMC, SPI, I²C, and IDMA channel operation. Table 18-8 shows the parameter RAM memory map.

Table 18-8. Parameter RAM Memory Map

Offset from IMMR	Page	Offset from DPRAM_base	Controller/Peripheral
0x3C00	1	0x1C00—0x1C7F	SCC1
		0x1C80—0x1CAF	I²C default area
		0x1CB0—0x1CBF	Miscellaneous
		0x1CC0—0x1CFF	IDMA1
0x3D00	2	0x1D00—0x1D7F	Reserved
		0x1D80—0x1DAF	SPI default area
		0x1DB0—0x1DBF	RISC timer table
		0x1DC0—0x1DFF	IDMA2

Table 18-8. Parameter RAM Memory Map (continued)

Offset from IMMR	Page	Offset from DPRAM_base	Controller/Peripheral
0x3E00	3	0x1E00—0x1E7F	Reserved
		0x1E80—0x1EBF	SMC1
		0x1EC0—0x1EFF	Reserved
0x3F00	4	0x1F00—0x1F7F	Reserved
		0x1F80—0x1FBF	SMC2/PIP
		0x1FC0—0x1FF	Reserved
		0x1FFE—0x1FFF	Reserved

The SPI and I^2C parameter RAM areas can be relocated to other 32-byte aligned parameter areas in dual-port RAM by programming their 16-bit base offsets, shown in Table 18-9.

Table 18-9. I^2C and SPI Parameter RAM Relocation

Offset from DPRAM_base	Size	Controller/Peripheral
0x1C80—0x1CAB	44 bytes	I^2C default parameter area
0x1CAC—0x1CAD	16 bits	I2C_BASE
0x1CAE—0x1CAF	16 bits	I^2C default parameter area
0x1D80—0x1DAB	44 bytes	SPI default parameter area
0x1DAC—0x1DAD	16 bits	SPI_BASE
0x1DAE—0x1DAF	16 bits	SPI default parameter area

The specific definition of each controllers' parameter RAM is protocol dependent and is described in the individual protocol chapters.

18.7 The RISC Timer Table

The CP can control a maximum of 16 timers separate and distinct from the four general-purpose timers and baud-rate generators of the CPM. The RISC timer table free the core from scanning a software timer table and are used for protocols that do not require extreme precision. The timers are clocked from an internal timer accessible to the CP only.

Each pair of timers can be configured as a pulse width modulation (PWM) channel; a maximum of eight channels are supportable. The output of the channel is driven on one of the port B pins.

The following list summarizes the main features of the RISC timer table:

- Supports up to 16 timers
- Supports up to 8 PWM channels
- Three timer modes: one-shot, restart, and PWM

- Maskable interrupt on timer expiration
- Programmable timer resolutions as low as 41 µs at 25 MHz
- Maximum timeout periods of 172 seconds at 25 MHz
- Continuously updated reference counter

RISC timer table operations are based on a “tick” in the CP internal timer that is programmed in the RCCR; see Section 18.5.1, “RISC Controller Configuration Register (RCCR).” The tick is a multiple of 1,024 general system clocks. The RISC timer table has the lowest priority of all CP operations, so if it is busy with other tasks and 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 CP; see Section 18.7.8, “Using the RISC Timers to Track CP Loading.” The timer table is configured using the RCCR, the timer table parameter RAM, and the RISC controller timer event/mask registers (RTER/RTMR), and by issuing SET TIMER to the CPCR.

18.7.1 RISC Timer Table Scan Algorithm

The CP scans the timer table once every tick of the internal CP timer. For each valid timer in the table, the CP decrements the count and checks for a timeout. If no timeout occurs, it moves to the next timer. If a timeout does occur, the CP sets the corresponding event bit in RTER and then checks R_TMR to see if the timer must be restarted. If it does, the CP leaves the timer valid bit set in the R_TMV register and resets the current count to the initial count; otherwise, the CP clears R_TMV. Once the timer table is scanned, the CP updates TM_CNT and stops working on the timer table until the next scan tick.

If SET TIMER is issued, the CP 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 CP timer. (Using SET TIMER properly synchronizes the timer table modifications to the execution of the CP.)

18.7.2 The SET TIMER Command

Issued to the CP command register (CPCR), the SET TIMER command is used to enable, disable, and configure the 16 timers in the RISC timer table. Set up the TM_CMD value in the RISC timer table parameter RAM before writing 0x0851 to the CPCR.

18.7.3 RISC Timer Table Parameter RAM and Timer Table Entries

Two areas of dual-port RAM are used for the RISC timer table—RISC timer table parameter RAM and the RISC timer table entries; see Figure 18-7.

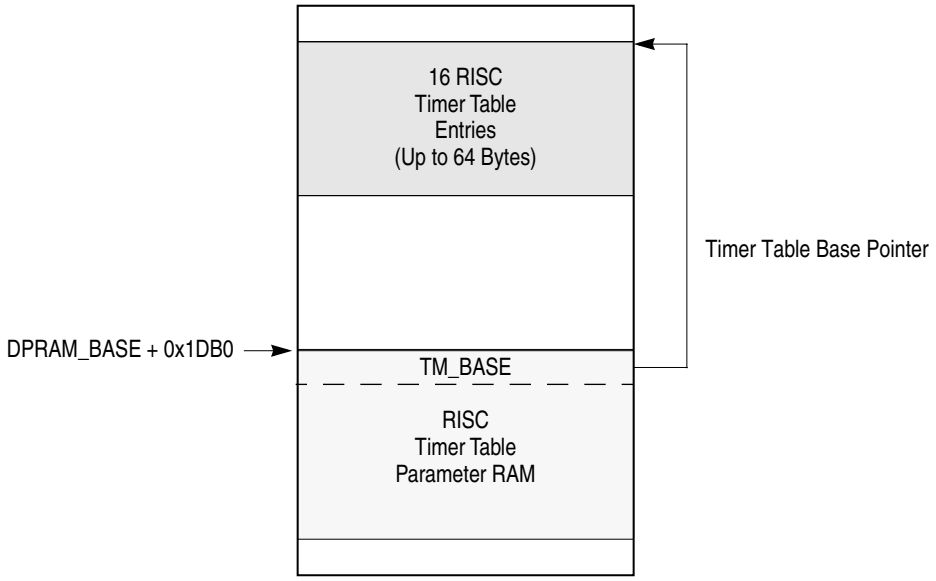


Figure 18-7. RISC Timer Table RAM Usage

The RISC timer table parameter RAM holds the general timer parameters. Table 18-10 shows its memory map.

Table 18-10. RISC Timer Table Parameter RAM Memory Map

Offset [1]	Name	Width	Description
0x00	**TM_BASE**	Hword	RISC timer table base address. The actual timers are a small block of memory in the dual-port RAM. TM_BASE is the offset from the beginning of the dual-port RAM where that block of memory resides. Four bytes must be reserved at the TM_BASE for each timer used, (64 bytes if all 16 timers are used). If fewer than 16 timers are used, timers should be allocated in ascending order to save space. For example, only 8 bytes are required if two timers are needed and RISC timers 0 and 1 are enabled. TM_BASE should be word-aligned.
0x02	TM_PTR	Hword	RISC timer table pointer. Only the CP uses this register to point to the next timer accessed in the timer table. Do not modify this register.
0x04	R_TMR	Hword	RISC timer mode register. Only the CP uses this register to store the mode of the timer, one-shot (0) or restart (1). Do not modify this register directly; it is modified indirectly via TM_CMD and the SET TIMER command.
0x06	R_TMV	Hword	RISC timer valid register. Only the CP uses this register to determine whether a timer is currently enabled. If the corresponding timer is enabled, a bit is 1. Do not modify this register directly; it is modified indirectly via TM_CMD and the SET TIMER command.
0x08	**TM_CMD**	Word	RISC timer command register. Used as a parameter location when SET TIMER is issued. Write this location prior to issuing SET TIMER. The bits of this register are defined in Section 18.7.3.1, “RISC Timer Command Register (TM_CMD)
0x0C	**TM_CNT**	Word	RISC timer internal count. Tick counter that the CP updates after each tick or after the timer table is scanned. It is updated if the CP’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 CP receives and responds to.

[1] From timer base address (IMMR + 3DB0)

18.7.3.1 RISC Timer Command Register (TM_CMD)

Figure 18-8 shows the TM_CMD register.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	V	R	PWM	—									Timer Number			
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	Timer Period															

Figure 18-8. RISC Timer Command Register (TM_CMD)

Table 18-11 describes TM_CMD fields.

Table 18-11. TM_CMD Field Descriptions

Bits	Name	Description
0	**V**	Valid. When set, this bit enables the timer. It should be cleared to disable the timer.
1	**R**	Restart. Should be set for an automatic restart or cleared for a one-shot timer operation.
2	**PWM**	Pulse width modulation mode. Set for PWM operation; see Section 18.7.5, “PWM Mode.”
3–11	—	Reserved. Should be cleared.
12–15	**Timer Number**	A value from 0–15 signifying which timer to use—an offset into the timer table entries.
16–31	**Timer Period**	The 16-bit timeout count of the timer. The minimum value is 1 and is programmed by writing 0x0000 to the timer period. The maximum value is 65,536 and is programmed by writing 0xFFFF.

18.7.3.2 RISC Timer Table Entries

The 16 timers are located in the block of memory pointed to by TM_BASE; each timer occupies 4 bytes. The first half-word holds the initial timer count taken from TM_CMD[Timer Period] when SET TIMER is executed; the next half-word is the current timer count that is decremented (until it reaches zero). Do not modify the table entries directly; instead, use the SET TIMER command to reinitialize table values.

18.7.4 RISC Timer Event Register (RTER)/Mask Register (RTMR)

The RISC timer event register (RTER), shown in Figure 18-9, reports period timeout events, which generate maskable interrupts. RTER bits are cleared by writing ones; writing zeros has no effect. This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

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	0000_0000_0000_0000															
R/W	R/W															
Addr	0x9D6 (RTER); 0x9DA (RTMR)															

Figure 18-9. RISC Timer Event Register (RTER)/Mask Register (RTMR)

The RISC timer mask register (RTMR), also shown in Figure 18-9, is used to enable interrupts generated in the RTER. If an individual timer's RTMR bit is set, the timer's RTER interrupt is enabled. If an RTMR bit is cleared, the corresponding interrupt in the RTER is masked. This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

The RISC timer table bit in the CPM interrupt mask register, CIMR[RTT], described in Section 34.5.3, "CPM Interrupt Mask Register," acts as a global RISC timer interrupt mask. Clearing CIMR[RTT] masks all RISC timer interrupts, regardless of RTMR.

18.7.5 PWM Mode

Designated pairs of timers can be used to generate PWM waveforms through port B. A maximum of eight channels are supported.

The first timer (even numbered) determines the duty cycle of the waveform:

- Program TM_CMD[Timer Period] to be the high period of the waveform.
- Set TM_CMD[V, PWM].

The second timer (odd numbered) determines the overall period:

- Program TM_CMD[Timer Period] to be the period of the whole waveform.
- Set TM_CMD[V, R] and clear TM_CMD[PWM].

Table 18-12 shows the port B pin assignments for PWM mode. The respective port B pins should be configured as general-purpose outputs; see Section 33.3, "Port B."

Table 18-12. PWM Channel Pin Assignments

Timer Pairs	Port B Pin
Timer 0, 1	Port B[23]
Timer 2, 3	Port B[22]
Timer 4, 5	Port B[21]
Timer 6, 7	Port B[20]
Timer 8, 9	Port B[19]
Timer 10, 11	Port B[18]

Table 18-12. PWM Channel Pin Assignments (continued)

Timer Pairs	Port B Pin
Timer 12, 13	Port B[17]
Timer 14, 15	Port B[16]

18.7.6 RISC Timer Initialization

Follow these steps to initialize the RISC timers:

1. Program RCCR[TIMEP] to the preferred internal timer tick interval, which determines the scan interval for the entire timer table. The timer enable bit, RCCR[TIME], is normally set 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 TM_BASE to point to a location in the dual-port RAM with $4 \times N$ bytes available, where N is the number of timers used. If N is less than 16, use timer 0 through timer N–1 to save space.
3. Clear TM_CNT to show how many ticks have elapsed since the CP internal timer was enabled (optional).
4. Clear the RTER, if it is not already cleared. Writing 0xFFFF clears this register.
5. Configure the RTMR to enable the timers that need to generate interrupts. A one enables interrupts.
6. Set CIMR[RTT] to generate interrupts to the system. The CPIC may require initialization not mentioned here; see Chapter 34, “CPM Interrupt Controller.”
7. Configure TM_CMD. 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 SET TIMER by writing 0x0851 to the CPCR.
9. Repeat the steps 7 and 8 for each timer to be enabled or disabled.

As an example, the following sequence demonstrates how RISC timer 0 is initialized to generate an interrupt approximately every second using a 25-MHz general system clock:

1. Write RCCR[TIMEP] with 0b111111 to generate the slowest timer. This value generates a table scan tick every 65,536 clocks, which is every 2.6 ms at 25 MHz.
2. Configure TM_BASE 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 TM_CNT to see how many ticks have elapsed since the CP 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 0x0002_0000 to the CPM interrupt mask register so the RISC timers will generate a system interrupt. Initialize the CPM interrupt configuration register.
7. Write 0xC000_0EE6 to TM_CMD. 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 SET TIMER.
9. Set RCCR[TIME] to start RISC timer table scanning.

18.7.7 RISC Timer Interrupt Handling

The following sequence shows what normally occurs within an interrupt handler for the RISC timer table:

1. Once an interrupt occurs, read the RTER to see which timers have caused interrupts. The RISC timer event bits are usually cleared at this time.
2. Issue any additional SET TIMER commands now or later, as preferred. Nothing needs to be done if the timer is automatically being restarted for repetitive interrupts.
3. Clear CISR[RTT].
4. Execute the **rfi** instruction.

18.7.8 Using the RISC Timers to Track CP Loading

The RISC timers can be used to track CP loading. The following sequence is a method for using the 16 RISC timers to determine if the CP ever exceeds the 96% utilization level during a scan tick interval. Removing the timers adds a 4% margin to the CP's utilization level, but an aggressive user can use this technique to push the CP performance to its limit. Incorporate the following steps to the standard initialization sequence:

1. Program RCCR[TIMEP] to 0b001111 for a table scan tick of $16 \times (1{,}024) = 16{,}384$.
2. Disable RISC timer table interrupts, if preferred.
3. Using SET TIMER, initialize all 16 RISC timers to have a timer period of 0xFFFF, which corresponds to 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 timer 15. If the difference between them exceeds two ticks, the CP has, during some scan tick interval, exceeded the 96% utilization level. Note that when comparing timer counts, the general-purpose timers are up-counters, while RISC timers are down-counters.

Chapter 19
SDMA Channels and IDMA Emulation

The CPM controls two physical serial DMA (SDMA) channels on the MPC855T. Using the two physical channels, the CP implements ten virtual SDMA channels, each dedicated to a serial controller transmitter or receiver—two for the full-duplex SCC, and the remaining eight for the SPI, I^2C, and the two SMCs. The CPM also emulates two general-purpose independent DMA (IDMA) channels for memory/memory and peripheral/memory transfers using the two physical SDMA channels.

19.1 SDMA Channels

Data from the SCC, SMCs, SPI, and I^2C can be routed to external memory (path 1) or the internal dual-port RAM (path 2), as shown in Figure 19-1. On a path 1 access, the SDMA channel must acquire both the U-bus and the external system bus. On a path 2 access, the data transfer occurs only on the U-bus, independent of the external system bus unless the SIU is configured in show-cycles mode. Thus, in normal operation, U-bus transfers occur simultaneously with external system bus operations.

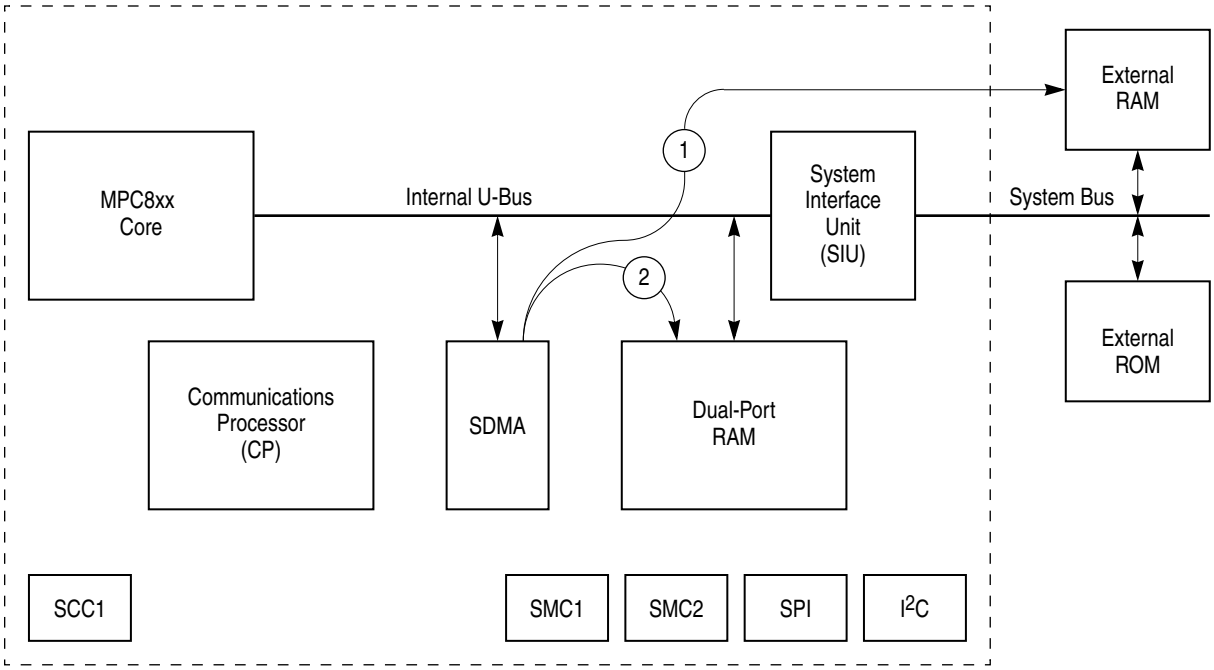


Figure 19-1. MPC855T SDMA Data Paths

19.1.1 SDMA Transfers

Each SDMA channel can be programmed to output a 3-bit function code that identifies the channel currently accessing memory. The SDMA channel can implement true little-endian, modified little-endian, or big-endian byte ordering when accessing buffers. These features are programmed in the receive and transmit function code registers associated with each serial controller and within an IDMA channel's BD; see Section 19.3.4, "IDMA Buffer Descriptors (BD)."

If a bus error occurs when the SDMA conducts an access, the CP generates a unique interrupt in the SDMA status register (SDSR). The interrupt service routine reads the SDMA address register (SDAR) to determine the address that the bus error occurred on. The individual channel that caused the bus error can be found by reading the Rx and Tx internal data pointers from the protocol-specific parameter RAM of the serial controllers. If an SDMA bus error occurs, all CPM activity ceases and the entire CPM must be reset in the CPM command register (CPCR); see Section 18.5.3, "CP Command Register (CPCR)."

19.1.2 U-Bus Arbitration and the SDMA Channels

The SDMA channels, I-cache, D-cache, and SIU all contend for the U-bus as internal masters with their relative priorities determined by an arbitration ID. The user can adjust the SDMA bus arbitration priority, but the other internal masters have fixed arbitration IDs; see Section 19.2.1, "SDMA Configuration Register (SDCR)." All 10 virtual SDMA channels share the same arbitration ID, and thus have the same priority relative to the other internal masters. See Table 19-1.

Table 19-1. U-Bus Arbitration IDs

Arbitration Level	Unit
7 (highest priority)	—
6	SDMA (SDCR[RAID]=0b00)
5	SDMA (SDCR[RAID]=0b01)
4	D-cache
3	I-cache
2	SDMA (SDCR[RAID]=0b10)
1	SDMA (SDCR[RAID]=0b11)
0	G2 core

Notes: DRAM refresh normally has a U-bus arbitration level of 0 (losing ties to the G2 core). However, if four refresh periods expire without servicing, the arbitration level is promoted to 7.
An external request loses ties to an internal request or DRAM refresh request with the same arbitration ID. For example, if SIUMCR[EARP] is 4, the external master has priority over the I-cache but not over the D-cache.

Once an SDMA channel obtains the external system bus, it remains master for the whole transaction—a byte, half-word, word or burst transfer—before relinquishing the bus. This feature, in combination with the zero-clock arbitration overhead provided by the U-bus, increases bus efficiency and lowers latency.

To minimize the latency associated with slower, character-oriented protocols, an SDMA writes each character to memory as it arrives without waiting for the next character, and always reads using 16-bit half-word transfers. A transfer may take multiple bus cycles if the memory provides a less than 32-bit port size. An SDMA uses back-to-back bus cycles for the entire transfer—4-word bursts, 32-bit reads, and 8-, 16-, or 32-bit writes—before relinquishing the bus. For example, an SDMA channel reading a 32-bit word from a 16-bit memory takes two consecutive bus cycles.

An SDMA steals cycles with no arbitration overhead unless an external device is bus master. Figure 19-2 shows an SDMA stealing a cycle from an internal bus master.

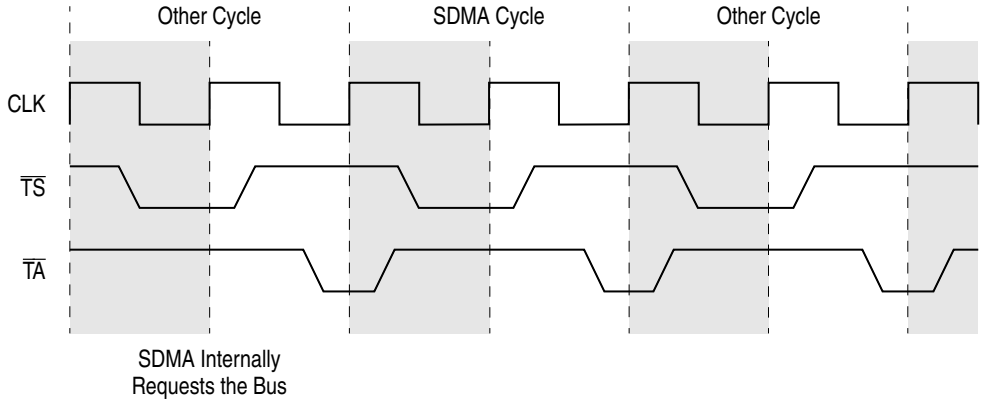


Figure 19-2. SDMA U-Bus Arbitration (Cycle Steal)

19.2 SDMA Registers

All SDMA channels share one configuration register (SDCR), a status register (SDSR), a mask register (SDMR), and a read-only, address register (SDAR). The configuration of each serial controller also affects their dedicated SDMA channels' behavior. The following sub-sections describe the SDMA registers.

19.2.1 SDMA Configuration Register (SDCR)

The SDMA configuration register (SDCR) configures all 10 virtual SDMA channels. It controls the channels' U-bus priority level and freeze-signal (FRZ) behavior. It is always read/write in supervisor mode, even though writing to the SDCR is not recommended unless the CPM is disabled. This register is affected by $\overline{\text{HRESET}}$ but is not affected by $\overline{\text{SRESET}}$. Figure 19-3 shows the register format.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Reset	0000_0000_0000_0000															
R/W	R															
Addr	IMMR + 0x030															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—	FRZ	—										FAID		RAID	
Reset	0000_0000_0000_0000															
R/W	R	R/W	R										R/W		R/W	
Addr	IMMR + 0x032															

Figure 19-3. SDMA Configuration Register (SDCR)

Table 19-2 describes the SDCR bit settings.

Table 19-2. SDCR Bit Settings

Bits	Name	Description
0–16	—	Reserved. Should be cleared.
17	FRZ	Freeze. Recognize or ignore the freeze signal. If configured to respond to the freeze signal, the SDMA controller negates $\overline{BR}$ until freeze is negated or a reset occurs. 0 SDMA channels ignore the freeze signal. 1 SDMA channels respond to a freeze on the next bus cycle.
18–27	—	Reserved. Should be cleared.
28–29	FAID	FEC arbitration ID. Determines FEC arbitration priority for the U bus; 00 for typical applications. 00 Priority 6 (highest) 01 Priority 5 10 Priority 2 11 Priority 1 (lowest)
30–31	RAID	RISC controller (CP) arbitration ID. Sets the SDMAs' U-bus arbitration priority level. Should be programmed to 0b01, priority level 5, for typical applications. (See Table 19-1 above.) 00 Priority level 6 (BR6). 01 Priority level 5 (BR5) (normal operation). 10 Priority level 2 (BR2). 11 Priority level 1 (BR1).

19.2.2 SDMA Status Register (SDSR)

Shared by all SDMA channels, the SDMA status register (SDSR) reports bus errors. When the SDMA controller recognizes an event, it sets the corresponding event bit in the SDSR. SDSR bits are cleared by writing ones; writing zeros has no effect. Figure 19-4 shows the register format.

Bit	0	1	2	3	4	5	6	7
Field	SBER	—						
Reset	0000_0000							
R/W	R/W							
Addr	IMMR + 0x908							

Figure 19-4. SDMA Status Register (SDSR)

This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Table 19-3 describes the SDSR bit settings.

Table 19-3. SDSR Field Descriptions

Bits	Name	Description
0	SBER	SDMA channel bus error. Indicates an error caused the SDMA channel to terminate during a read or write cycle. The SDMA bus error address can be retrieved from the SDMA address register (SDAR).
1–7	—	Reserved

19.2.3 SDMA Mask Register (SDMR)

The read/write SDMA mask register (SDMR) has the same bit format as SDSR; see above Figure 19-4. If a bit in the SDMR is set, the corresponding interrupt in the SDSR is enabled; if the bit is cleared, the corresponding interrupt is masked. Reset clears SDMR. Its internal address (IMMR offset) is 0x90C. This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

19.2.4 SDMA Address Register (SDAR)

The 32-bit, read-only SDMA address register (SDAR) holds the current system address being accessed and is used to diagnose an SDMA bus error. SDAR is undefined at reset. Its internal address (IMMR offset) is 0x904. This register is not affected by $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$.

19.3 IDMA Emulation

The CPM can be configured to emulate two general-purpose independent DMA (IDMA) channels using the two physical SDMA channels. In IDMA emulation mode, the user specifies a memory/memory or peripheral/memory transfer as if using dedicated DMA hardware.

IDMA uses two addressing modes—dual-address and single-address. In IDMA dual-address transfers, data is read into internal storage, packed onto the bus, and then written to the destination. Dual-address transfers can take several bus cycles depending on the peripheral's port size. In contrast, single-address (fly-by) IDMA bypasses internal storage, transferring data directly between memory and a peripheral in a single bus cycle. See Section 19.3.8, "IDMA Transfers—Dual-Address and Single-Address."

The IDMA controller supports two buffer handling modes—auto-buffering, and buffer-chaining. In buffer-chaining, an IDMA moves a connected series of BDs called a chain without interruption. Auto-buffering allows a buffer chain to be repeatedly transferred in a loop without user intervention. See Section 19.3.4.2, “Auto-Buffering and Buffer-Chaining.”

Single-buffering is a special, low-latency IDMA transfer mode optimized for transferring one buffer from a peripheral to memory. This low-overhead mode uses single-address (fly-by) burst transfers. Note that single-buffering is available only on IDMA1. This mode also remaps the IDMA1 channel parameter RAM. See Section 19.3.9, “Single-Buffer Mode on IDMA1—A Special Case.”

Note that $\overline{\text{DREQ0}}$ is the DMA request for IDMA1, and $\overline{\text{DREQ1}}$ is the DMA request for IDMA2.

19.3.1 IDMA Features

The following is a list of 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 allow for 4-Gbyte buffers
- Byte, half-word, word, or 4-word burst quantities for transfers
- 32-bit, byte-addressable buffer pointers auto-increment for memory accesses and remain constant for peripheral accesses
- Byte-packing and unpacking algorithms use the absolute minimum number of bus cycles required during dual-address transfers
- All bus-termination modes, such as $\overline{\text{TA}}$, $\overline{\text{TEA}}$, and $\overline{\text{BI}}$, are supported
- DMA handshaking for cycle-steal and burst transfers
- Two buffer handling modes—auto-buffering and buffer-chaining
- Optimized, low-overhead single-buffer mode for peripheral-to-memory transfers on IDMA1
- The MPC855T’s chip-select and wait-state generation logic can be used with IDMA.

19.3.2 IDMA Parameter RAM

Both IDMA channels have a dedicated portion of dual-port RAM for channel parameters. Table 19-4 shows the memory map. Note that in the special single-buffer mode, the IDMA1 parameter RAM map changes; see Section 19.3.9, “Single-Buffer Mode on IDMA1—A Special Case.”

Table 19-4. IDMA Parameter RAM Memory Map

Offset [1]	Name	Width	Description
0x00	**IBASE**	Hword	IDMA BD base address. Defines the base address of the area in dual-port RAM set aside for this channel's BD table. It is an offset from the beginning of dual-port RAM. Note that IBASE should be burst-aligned (divisible by 16).
0x02	**DCMR**	Hword	DMA channel mode register. See Section 19.3.3.1, "DMA Channel Mode Registers (DCMR)."
0x04	SAPR	Word	Source data pointer (internal-use). Points to the next source byte to be read.The CP initializes SAPR to the BD's source buffer pointer and increments it automatically if the source is memory (DCMR[S/D] = 0bx0).
0x08	DAPR	Word	Destination data pointer (internal-use). Points to the next destination byte to be written. The CP initializes DAPR to the BD's destination buffer pointer, and increments it automatically if the destination is memory (DCMR[S/D] = 0b0x).
0x0C	IBPTR	Hword	Current IDMA BD pointer. If the IDMA channel is idle, IBPTR points to the next valid BD in the table. After a reset, or when the end (wrap bit) of the BD table is reached, the CP wraps IBPTR back to IBASE.
0x0E	WRITE_SP	Hword	Internal-use
0x10	S_BYTE_C	Word	Internal source byte count
0x14	D_BYTE_C	Word	Internal destination byte count
0x18	S_STATE	Word	Internal state
0x1C	ITEMP	4 Words	Temp data storage
0x2C	SR_MEM	Word	Data storage for peripheral write
0x30	READ_SP	Hword	Internal-use
0x32	—	Hword	Difference between source and destination residue
0x34	—	Hword	Temp storage address pointer
0x36	—	Hword	SR_MEM byte count
0x38	D_STATE	Word	Reserved. Internal state used by CP
• Notes: Boldfaced items must be initialized by the user before enabling an IDMA channel. The remaining parameters are used by the CP only.			

1 IDMA1 base = IMMR + 0x3CC0
IDMA2 base = IMMR + 0x3DC0

19.3.3 IDMA Registers

Each IDMA channel has a DMA channel mode register (DCMR), an IDMA status register (IDSR) and corresponding mask register (IDMR) that contain global channel parameters.

19.3.3.1 DMA Channel Mode Registers (DCMR)

Located in each IDMA's parameter RAM, the DMA channel mode registers (DCMR) configure the peripheral port size, the source and destination type of the transfer, and the address mode (cycle mode) of the IDMA channels. Figure 19-5 shows the register format.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—											**SIZE**		**S/D**		**SC**
Reset	0											0		0		0
R/W	R											R/W		R/W		R/W
Addr	IDMA*x* Base + 0x02															

Figure 19-5. DMA Channel Mode Register (DCMR)

Table 19-5 describes DCMR fields.

Table 19-5. DCMR Field Descriptions

Bits	Name	Description
0–10	—	Reserved. Should be cleared.
11–12	**SIZE**	Peripheral port size. Determines the operand transfer size per $\overline{\text{DREQ}}x$ assertion for peripheral/memory transfers, but not for memory/memory transfers. (For memory/memory transfers the size is determined only by address alignment and the amount of data remaining to be transferred.) 00 Word length. 01 Half-word length. 10 Byte length. 11 Reserved. Note that the memory port size is transparent to the IDMA. The SIU emulates a 32-bit port size regardless of the actual memory port size.
13–14	**S/D**	Source/destination. Defines the source and destination—memory or peripheral. 00 Read from memory; write to memory. 01 Read from peripheral; write to memory. 10 Read from memory; write to peripheral. 11 Reserved. Note that for memory/memory accesses, the CP automatically increments the address and does not use $\overline{\text{SDACK}}n$.
15	**SC**	Single-cycle. Selects single- or dual-cycle mode. 0 Dual-cycle (dual-address) mode. 1 Single-cycle (single-address) mode.

19.3.3.2 IDMA Status Registers (IDSR1 and IDSR2)

The IDMA status registers (IDSR1 and IDSR2) report transfer events. When the IDMA controller recognizes an event, it sets the corresponding event bit in the IDSR. IDSR bits are cleared by writing ones; writing zeros has no effect. Figure 19-6 shows the register format.

Bit	0	1	2	3	4	5	6	7
Field	—					AD	DONE	OB
Reset	0					0	0	0
R/W	R					R/W	R/W	R/W
Addr	IMMR + 0x910 (IDSR1); 0x918 (IDSR2)							

Figure 19-6. IDMA Status Registers (IDSR1/IDSR2)

This registers are affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Table 19-6 describes the IDSR fields.

Table 19-6. IDSR1/IDSR2 Field Descriptions

Bits	Name	Description
0–4	—	Reserved
5	AD	Auxiliary done. Set after processing a BD that has its I bit (interrupt) set.
6	DONE	Buffer chain done. Indicates IDMA transfer termination. Set after servicing a BD that has its L bit (last) set, regardless of the I bit setting.
7	OB	Out of buffers. Indicates that the IDMA channel has no valid BDs left in the BD table.

19.3.3.3 IDMA Mask Registers (IDMR1 and IDMR2)

The read/write IDMA mask registers (IDMR1 and IDMR2) have the same format as IDSR, shown in Figure 19-6. If an IDMR bit is set, the corresponding interrupt is enabled in IDSR*n*; if it is cleared, the corresponding interrupt is masked. Reset clears IDMR. IDMR1's internal address (IMMR offset) is 0x914; IDMR2's is 0x91C. These registers are affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

19.3.4 IDMA Buffer Descriptors (BD)

An IDMA buffer descriptor contains the specific transfer information needed for its buffer. IDMA BDs contain a status-and-control field, the function code registers, the buffer length, and the source and destination buffer pointers. The BDs are grouped together in contiguous dual-port RAM to form a standard BD table; see Figure 19-7.

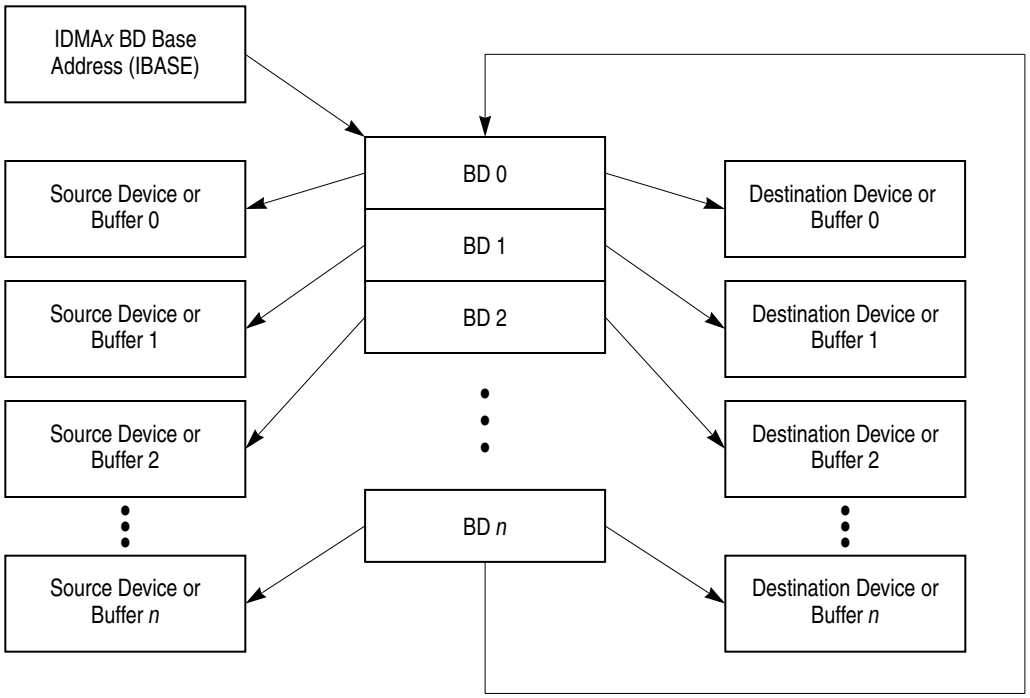


Figure 19-7. IDMA*x* Channel's BD Table

An IDMA descriptor breaks down as follows:

- The half word at (offset + 0) is the status-and-control field.
- The byte at (offset + 2) is the destination function code register (DFCR). See Section 19.3.4.1, "Function Code Registers—SFCR and DFCR."
- The byte at (offset + 3) is the source function code register (SFCR). See Section 19.3.4.1, "Function Code Registers—SFCR and DFCR."
- The word at (offset + 4) is the buffer length, containing the number of bytes for transfer. It must be greater than zero.
- The word at (offset + 8) points to the beginning of the source buffer in internal or external memory.
  - When the source is a peripheral, this field is ignored in single-address mode. In dual-address mode, this field contains the peripheral address.
- The word at (offset + 0xC) points to the beginning of the destination buffer in internal or external memory.
  - When the destination is a peripheral, this field is ignored in single-address mode. In dual-address mode, this field contains the peripheral address.

Figure 19-8 shows the descriptor structure.

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x00	**V**	—	**W**	**I**	**L**	—	**CM**	—	—	—	—	—	—	—	—	—
0x02	**DFCR**								**SFCR**							
0x04	**Buffer Length**															
0x06																
0x08	**Source Buffer Pointer**															
0x0A																
0x0C	**Destination Buffer Pointer**															
0x0E																

Figure 19-8. IDMA Buffer Descriptor Structure

Table 19-7 describes an IDMA descriptor's status-and-control field.

Table 19-7. IDMA BD Status and Control Bits

Bits	Name	Description
0	**V**	Valid. Ready for processing. 0 Invalid. Not ready for transfer. The user can write to this descriptor and its buffer. When buffer-chaining, the CP clears the V bit after the buffer has been transferred. 1 Valid for transfer. The user should not write to this descriptor or its buffer once the V bit is set. Note: When an error condition is detected, the CP clears the V bit.
1	—	Reserved
2	**W**	Wrap. Marks the end of the BD table. 0 Not the last descriptor in the BD table. 1 Last descriptor in the BD table. After this descriptor has been processed, the CP wraps the current BD pointer (IBPTR) back to the top of the BD table (IBASE).
3	**I**	Interrupt. Enable the maskable auxiliary-done (AD) interrupt. 0 IDSR[AD] is not flagged after this BD is processed. 1 IDSR[AD] is flagged after this BD is processed.
4	**L**	Last. Marks the end of a buffer chain and enables the maskable DONE interrupt. 0 Not the last BD of a buffer chain. 1 Last BD of a buffer chain. When the transfer count is exhausted, IDSR[DONE] is flagged, regardless of the I bit.
5	—	Reserved
6	**CM**	Continuous mode. Selects buffer-chaining or auto-buffering; see Section 19.3.4.2, "Auto-Buffering and Buffer-Chaining." 0 Normal mode (buffer-chaining). The CP clears the V bit after this descriptor is processed. 1 Continuous mode (auto-buffering). The CP does not clear the V bit after this descriptor is processed.
7–15	—	Reserved

19.3.4.1 Function Code Registers—SFCR and DFCR

The user programs an IDMA channel's source and destination function code registers (SFCR and DFCR) with separate 3-bit function codes to tag the channel's source and destination accesses. The function code registers also determine the byte-ordering convention. Figure 19-9 shows the register format.

Bit	0	1	2	3	4	5	6	7
Field	—			BO		AT[1–3]		
Addr	DFCR is at offset 0x02. SFCR is at offset 0x03.							

Figure 19-9. Function Code Registers—SFCR and DFCR

Table 19-8 describes the function code register bit settings.

Table 19-8. SFCR and DFCR Field Descriptions

Bits	Name	Description
0–2	—	Reserved. Should be cleared.
3–4	**BO**	Byte ordering. Set BO to select the required byte ordering for the buffer. If BO is changed on-the-fly, it takes effect at the beginning of the next frame (Ethernet, HDLC, and transparent) or at the beginning of the next BD. See Appendix A, "Byte Ordering." 00 Reserved 01 Modified little-endian. 1x Big-endian or true little-endian.
5–7	**AT[1–3]**	Address type 1–3. Holds the function code for an IDMA channel memory access. Note AT[0] is driven high to identify the access as a DMA type. Note that for the last IDMA cycle, the terminal count code AT[0–3] = 0xF replaces the user-defined function code signaling the end of transfer to the peripheral.

19.3.4.2 Auto-Buffering and Buffer-Chaining

Buffer-chaining is designed to move large amounts of noncontiguous blocks of data. Even though each block needs a separate BD, the BDs can be chained together and serviced as a group. Auto-buffering is used to repeatedly service a BD chain. Note that a chain can range from one BD to the whole BD table in length.

Setting the CM bit (continuous mode) in a BD's status-and-control field enables auto-buffering; clearing the CM bit enables buffer-chaining (normal mode). The CM bit must be explicitly programmed for each BD.

When auto-buffering, the descriptor's V bit will not be cleared after CPM processing—the BD remains valid for immediate transfer as the current BD pointer cycles through the table. When buffer-chaining, the CPM invalidates the current BD after processing to allow the user (the core) to safely manipulate the contents of the buffer and modify its BD. Note that the V bit behavior is the only difference between auto-buffering and buffer-chaining—auto-buffering can be thought of as continuous buffer-chaining. One use

of auto-buffering is for continuous monitoring of an external instrument, such as an A/D converter.

Set the L bit (last) in the status-and-control field to mark the last BD of a chain. When the CPM completes a chain, it flags IDSR[DONE], triggering a maskable interrupt to the core. The I bit (individual BD interrupt) behavior is independent of the L bit—the user may disable individual BD interrupts (and/or mask them) for multi-buffer chains.

19.3.5 IDMA CP Commands

The core issues the following IDMA commands to the CP:

- INIT IDMA—The CPM resets the IDMA internal state. The current BD pointer is reset to the top of the BD table (IBASE).
- STOP IDMA—The CP terminates current IDMA transfers. IDSR[DONE] is set, and the current BD is closed. If the destination is memory, the IDMA internal storage buffer is transferred before termination, regardless of the source. If the destination is a peripheral, the internal storage buffer is flushed and the transfer terminated without writing to the peripheral. At the next request, the next BD in the table is processed.

See Section 18.5.3, “CP Command Register (CPCR),” for the mechanics of issuing CP commands.

19.3.6 IDMA Channel Operation

An IDMA channel operation consists of the following events—IDMA channel initialization, data transfer, and block termination. In the initialization phase, the core loads the global IDMA channel information into the IDMA parameter RAM, builds the IDMA BD table, and starts the channel. In the transfer phase, the CPM accepts a transfer request, reads the transfer-specific information from the current BD into the IDMA parameter RAM, programs the physical SDMA channel, and provides addressing and bus control. The termination phase begins when the transfer byte count reaches zero (or a bus error occurs). The CPM then interrupts the core (unless masked), and the current BD pointer moves to the next BD in the table.

To begin a block transfer, initialize the IDMA registers, and build the IDMA BDs with information describing the data block, device type, and other special control options. See Section 19.3.2, “IDMA Parameter RAM,” and Section 19.3.5, “IDMA CP Commands.”

19.3.6.1 Activating an IDMA Channel

IDMA requests are generated externally via the $\overline{\text{DREQ}}$ signals. (There is no mechanism for generating internal IDMA requests.) After initializing the IDMA parameter RAM and the BD table, enable the $\overline{\text{DREQ}}$ signal by setting the corresponding PCSO[DREQ] of the port C special options register; see Section 33.4.2.4, “Port C Special Options Register

(PCSO).” Enabling the DREQ signal effectively activates the corresponding IDMA channel. Requests for IDMA1 have priority over IDMA2.

19.3.6.2 Suspending an IDMA Channel

Disabling the corresponding DREQ signal by clearing the corresponding PCSO[DREQ] suspends the IDMA channel transfer. A transfer in progress will be completed before the bus is released. No further bus cycles are started while PCSO[DREQ] remains cleared. During channel suspension, the core can access IDMA internal registers to determine the status of the channel or to alter parameters. If PCSO[DREQ] is set again while a transfer request is pending, the channel arbitrates for the bus and continues normal operation.

19.3.7IDMA Interface Signals—DREQ and SDACK

Each IDMA channel (IDMA1 and IDMA2) has two dedicated control signals—DMA request (DREQ) and SDMA acknowledge (SDACK). DREQ0 and SDACK1 are dedicated to IDMA1, while DREQ1 and SDACK2 are for IDMA2.

DREQ and SDACK are the handshake signals between the MPC855T and an external peripheral requesting service. A peripheral requests IDMA service directly to the CPM by asserting DREQ. When the CPM begins the transfer, it acknowledges the peripheral by asserting SDACK. A requesting peripheral can either be the source or the destination of an IDMA transfer. Note that SDACK is not used for memory/memory transfers.

The following subsections discuss the interface signals used for requesting memory/memory and peripheral/memory transfers.

19.3.7.1 IDMA Requests for Memory/Memory Transfers

Because there is no internal mechanism, an externally-connected DREQ must still be used to generate IDMA memory/memory transfer requests. This can be done using a general-purpose I/O line or a general-purpose timer output.

To use a general-purpose I/O line, follow these steps:

1. Externally connect a general-purpose output line to DREQ.
2. Set RCCR[DR*n*M] (level-sensitive).
3. Drive the output low when the request generation should begin.

The IDMA controller continuously requests the bus until the current buffer chain is completely transferred. The transfer terminates with an out-of-buffers error (IDSR[OB]).

To use a general-purpose timer output (TOUT*x*), follow these steps:

1. Externally connect a TOUT*x* to DREQ.
2. Clear RCCR[DR*n*M] (edge-sensitive).
3. Program the timer period to pace the IDMA requests (and thus bus utilization).

An interrupt handler can service the IDSR[DONE] interrupt and suspend the channel; otherwise, the transfer terminates with an out-of-buffers error (IDSR[OB]).

19.3.7.2 IDMA Requests for Peripheral/Memory Transfers

Once an IDMA channel has been activated, an external peripheral requests a transfer using $\overline{\text{DREQ}}$. The user programs the RISC controller (the CP) configuration register (RCCR) to make IDMA requests either edge- or level-sensitive. The RCCR settings also determine the priority of $\overline{\text{DREQ}}$ relative to the SCC. See Section 18.5.1, “RISC Controller Configuration Register (RCCR).” Since $\overline{\text{DREQ0}}$ and $\overline{\text{DREQ1}}$ are multiplexed through PC15 and PC14 respectively, the port C pin assignment register and direction register must be configured as well; see Section 33.4.2, “Port C Registers.”

Level-sensitive mode maximizes IDMA channel bandwidth for peripherals requiring high transfer rates. For external peripherals that generate a pulsed transfer signal for each data operand, edge-sensitive requests should be used.

19.3.7.2.1 Level-Sensitive Requests

Setting RCCR[DR*n*M] makes the corresponding IDMA channel level-sensitive to requests. $\overline{\text{DREQ}}$ is sampled at rising edge of the clock. The peripheral requests service by asserting $\overline{\text{DREQ}}$ and leaving it asserted as long as it needs service.

Each time the IDMA controller issues a bus cycle either to read or write the peripheral, it asserts $\overline{\text{SDACK}}$ to acknowledge the original transfer request on $\overline{\text{DREQ}}$. The IDMA channel continues moving data in back-to-back DMA cycles until $\overline{\text{DREQ}}$ is negated. To ensure the correct number of DMA transfers are performed, the peripheral must negate $\overline{\text{DREQ}}$ while the IDMA is acknowledging the last data move, that is, while $\overline{\text{SDACK}}$ is asserted. $\overline{\text{DREQ}}$ is sampled on the same rising edge on which $\overline{\text{TA}}$ is sampled to terminate the current cycle.

19.3.7.2.2 Edge-Sensitive Requests

Clearing RCCR[DR*n*M] makes the corresponding IDMA channel edge-sensitive to requests. The edge sensitivity is further qualified to detect either any edge or falling edges only as programmed in PCINT[EDM15] and PCINT[EDM14] for $\overline{\text{DREQ0}}$ and $\overline{\text{DREQ1}}$, respectively; see Section Figure 33-15., “Port C Interrupt Control Register (PCINT).”

In edge-sensitive mode, an IDMA channel moves one data operand per request. $\overline{\text{DREQ}}$ is sampled at each rising edge of the clock. When IDMA detects a request on $\overline{\text{DREQ}}$, the request is considered pending and remains pending until it is processed. Subsequent requests on $\overline{\text{DREQ}}$ are ignored until the pending request is acknowledged.

19.3.8 IDMA Transfers—Dual-Address and Single-Address

Once an IDMA channel successfully arbitrates for the bus, it begins the transfer. An IDMA channel has the same bus cycle timing as the other internal masters.

The IDMA controller supports both dual- and single-address transfers. The dual-address transfer consists of a source read and a destination write—a memory/memory or memory/peripheral transfer. A single-address transfer, also called fly-by, consists of one external read or write bus cycle—a memory/peripheral transfer.

19.3.8.1 Dual-Address (Dual-Cycle) Transfer

The IDMA channels can operate in a dual-address transfer mode in which data is first read using the source pointer and placed in internal storage. The data operand is then packed onto the bus and written to the address given by the destination pointer. The read and write transfers can take several bus cycles each because of differences in the source and destination operand sizes. The dual-address read and write cycles are described below.

- Dual-address source read—SAPR drives the address bus, SFCR drives the address type, and DCMR drives the size control. Data is read from the memory or peripheral and placed in internal storage at the end of the bus cycle. For memory reads, SAPR is automatically incremented by 1, 2, 4, or 16, depending on the address and size information specified by DCMR. See Section 19.3.2, “IDMA Parameter RAM,” and Section 19.3.3.1, “DMA Channel Mode Registers (DCMR).”
- Dual-address destination write—The data in internal storage is written to the peripheral or memory governed by the address in DAPR, the address type in DFCR, and the size in DCMR. For memory writes, DAPR is automatically incremented by 1, 2, 4, or 16 according to DCMR. The byte count is decremented by the number of bytes transferred. When the byte count reaches zero and the transfer reports no errors, IDSR[DONE] is flagged, which triggers a maskable interrupt. See Section 19.3.2, “IDMA Parameter RAM,” and Section 19.3.3, “IDMA Registers.”

Additionally, for peripheral/memory dual-address transfers, the $\overline{\text{SDACK}}$ signal asserts during the peripheral access. For dual-address transfers, microcode performs byte-packing using a 16-byte buffer in the dual-port RAM. 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 least number of bus cycles.

19.3.8.2 Single-Address (Single-Cycle) Transfer (Fly-By)

Each IDMA channel can be independently programmed to provide single-address, or fly-by, transfers. The IDMA channel bypasses or flies-by internal storage since the transfer occurs directly between a device and memory. DCMR[S/D] controls the direction of the transfer. If DCMR[S/D] = 0b01, the IDMA controller handshakes with the peripheral for the source data and writes to the destination memory address provided. If DCMR[S/D] = 0b10, the IDMA controller handshakes with the destination peripheral and reads from the source memory address provided. The single-address read and write cycles are described below.

- Single-address memory-read/peripheral-write—The memory address in SAPR, the address type in SFCR, and the size in DCMR provide the data and control signals to the data bus. This bus cycle operates like a normal read bus cycle. The SAPR is incremented by 1, 2, or 4, according to the programming of DCMR[SIZE]. The destination device is controlled by the IDMA handshake signals $\overline{\text{DREQ}}$ and $\overline{\text{SDACK}}$. Asserting $\overline{\text{SDACK}}$ provides write control to the destination device. Figure 19-10 and Figure 19-11 show the transaction timing diagrams for asynchronous and synchronous single-address peripheral writes. See Section 19.3.7, “IDMA Interface Signals—DREQ and SDACK,” for more on IDMA handshake signals.

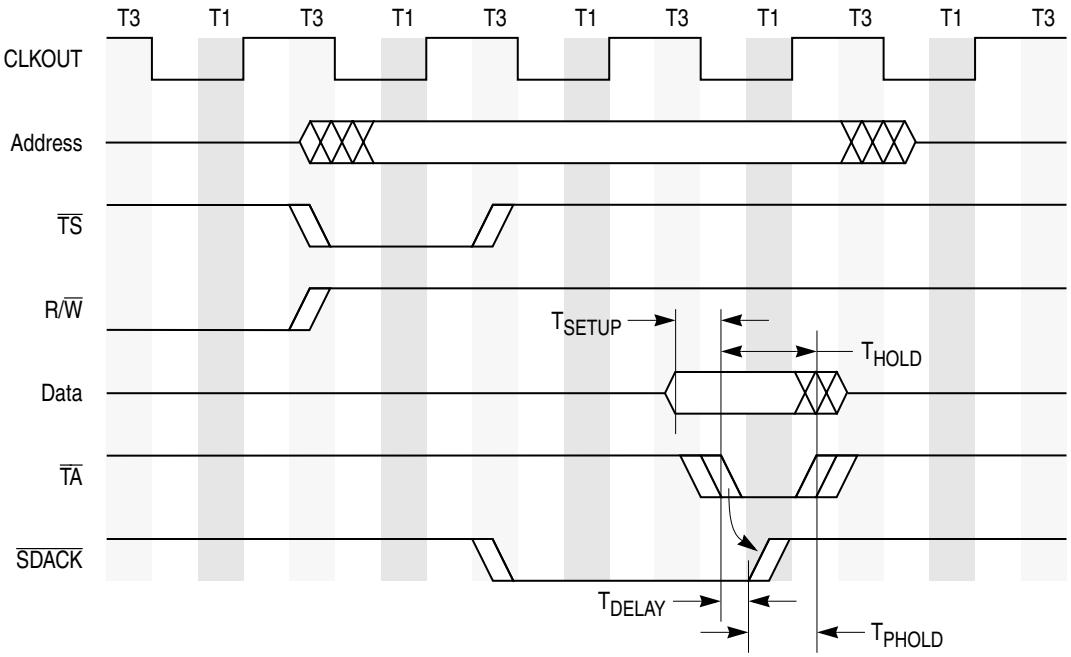


Figure 19-10. $\overline{\text{SDACK}}$ Timing Diagram: Single-Address Peripheral Write, Externally-Generated $\overline{\text{TA}}$

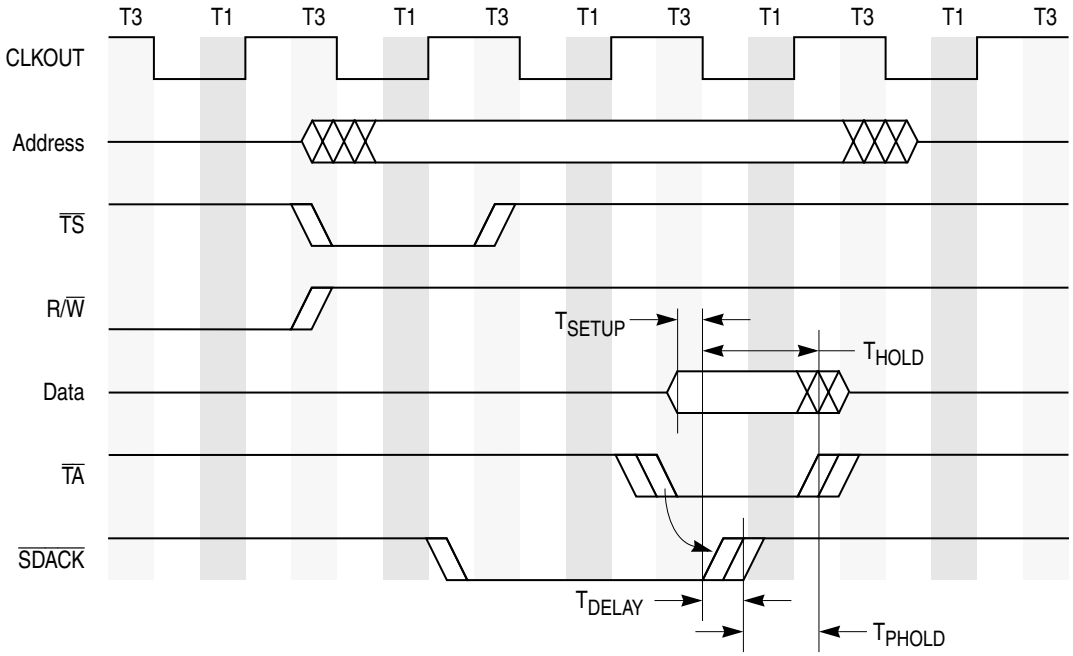


Figure 19-11. SDACK Timing Diagram: Single-Address Peripheral Write, Internally-Generated TA

- Single-address memory-write/peripheral-read—The source device is controlled by the IDMA handshake signals (DREQ and SDACK). When the source device requests service from the IDMA channel, IDMA asserts SDACK to allow the source device to drive data onto the data bus. The data is written to the memory address in DAPR, the address type in DFCR, and the size in DCMR. The data bus is three-stated for this write cycle. The DAPR is incremented by 1, 2, or 4, according to the programming of DCMR[SIZE]. See Section 19.3.7, “IDMA Interface Signals—DREQ and SDACK,” for more on IDMA handshake signals.

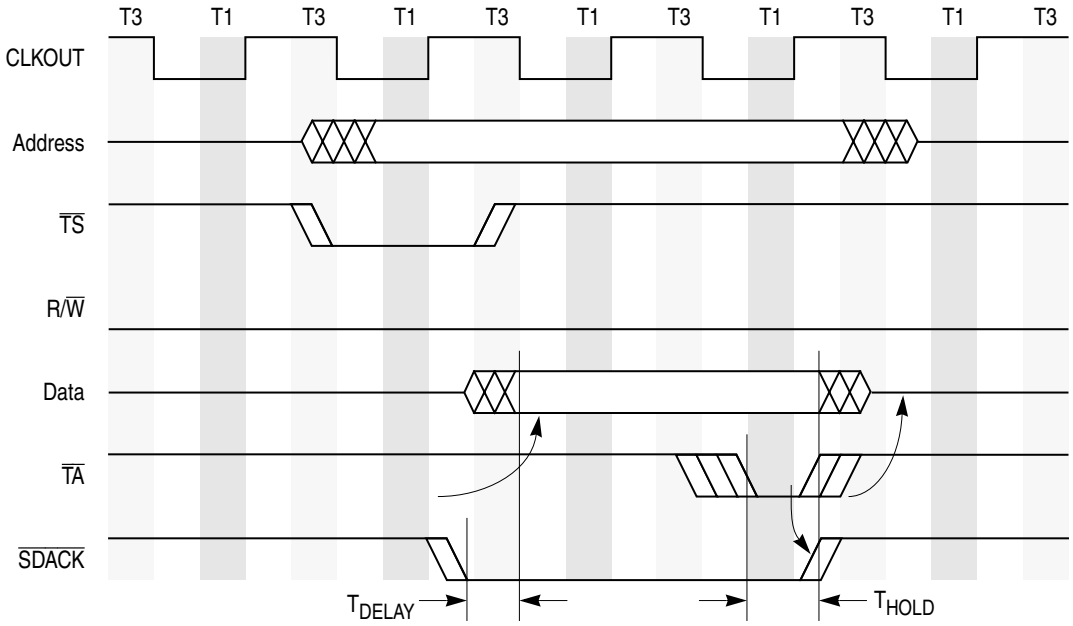


Figure 19-12. SDACK Timing Diagram: Single-Address Peripheral Read, Internally-Generated TA

19.3.9 Single-Buffer Mode on IDMA1—A Special Case

For single-buffer transfers from a peripheral to memory of up to 64 bytes per request, IDMA1 offers a reduced-latency solution using single-address bursts. The memory destination address, the buffer length (byte count), and the channel mode register are stored directly in the IDMA parameter RAM instead of in a formal BD. Table 19-9 shows the single-buffer mode IDMA1 parameter RAM map.

Table 19-9. Single-Buffer Mode IDMA1 Parameter RAM Map

Offset [1]	Name	Width	Description
0x00	**BAPR**	Word	Buffer pointer. Contains the destination buffer memory address. BAPR should be burst-aligned. It is automatically incremented by 16 bytes after each burst.
0x04	**BCR**	Word	Byte count register. Contains the buffer length in bytes. BCR is decremented by 16 after each burst. BCR must be a multiple of 16. The IDMA channel will terminate the block transfer when BCR reaches zero.
0x08	**DCMR**	Word	DMA channel mode register.
0x0C–0x3F	—	—	Reserved.

Note: Parameters should not be modified while DMA is active.

1 From IDMA1 base = IMMR + 0x3CC0

Single-buffer mode is selected by setting RCCR[EIE], the CPM external interrupt enable bit; see Section 18.5.1, “RISC Controller Configuration Register (RCCR).” Note that the CPM external interrupt always refers to a special request to the CPM, not to the core.

19.3.9.1 IDMA1 Channel Mode Register (DCMR) (Single-Buffer Mode)

DCMR contains the channel’s function code and byte-order convention, previously held in function code registers. DCMR also holds the channel start bit (enable) and burst transfer information. Figure 19-13 shows the DCMR format.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	**1**	—		**BO**		**AT[1–3]**			**STR**	—					**BPR**	
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	IDMA1 Base + 0x08															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—															
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	IDMA1 Base + 0x0A															

Figure 19-13. IDMA Channel Mode Register (DCMR) (Single-Buffer Mode)

Table 19-10 describes the DCMR bit settings for IDMA1 in single-buffer mode.

Table 19-10. DCMR Field Descriptions (Single-Buffer Mode)

Bits	Name	Description
0	—	Must be set.
1–2	—	Reserved. Should be cleared.

Table 19-10. DCMR Field Descriptions (Single-Buffer Mode) (continued)

Bits	Name	Description
3–4	**BO**	See corresponding description in Table 19-8 above.
5–7	**AT[1–3]**	See corresponding description in Table 19-8 above.
8	**STR**	Start. Enables the IDMA channel. Cleared automatically upon completion of the transfer request. 0 Disable IDMA channel. 1 Enable IDMA channel. Set STR after programming BAPR and BCR.
9–13	—	Reserved. Should be cleared.
14–15	**BPR**	Bursts per request. Determines how many bursts will be initiated for each request. 00 One burst per request (16 bytes). 01 Two bursts per request (32 bytes). 10 Reserved 11 Four bursts per request (64 bytes).
16–31	—	Reserved. Should be cleared.

19.3.9.2 IDMA1 Status Register (IDSR1) (Single-Buffer Mode)

IDSR1 in single-buffer mode behaves the same way as defined in Section 19.3.3.2, “IDMA Status Registers (IDSR1 and IDSR2).” The only relevant event bit, however, is DONE, which is set when the byte count in BCR reaches zero. Figure 19-14 shows the register format.

Bits	0	1	2	3	4	5	6	7
Field	—						DONE	—
Reset	0000_0000_0000_0000							
R/W	R/W							
Addr	IMMR + 0x910							

Figure 19-14. IDMA1 Status Register (IDSR1) (Single-Buffer Mode)

19.3.9.3 IDMA1 Mask Register (IDMR1) (Single-Buffer Mode)

IDMR1 in single-buffer mode behaves the same way as defined above; see Section 19.3.3.3, “IDMA Mask Registers (IDMR1 and IDMR2).” Figure 19-14 above shows the mask register’s format in single-buffer mode. IDMR1’s internal address (IMMR offset) is 0x914.

19.3.9.4 Burst Timing (Single-Buffer Mode)

A typical single-address burst timing when IDMA1 is in single-buffer mode, is illustrated in Figure 19-15. The peripheral asserts $\overline{\text{DREQ0}}$ and waits for $\overline{\text{SDACK1}}$ to initiate a burst transfer to memory. The peripheral must negate $\overline{\text{DREQ0}}$ before the last beat of the transfer; otherwise, IDMA assumes that another DMA request is pending—DCMR[STR] will not

be cleared—and immediately initiates another transfer. If no buffer is available when this extra transfer begins, erratic operation occurs.

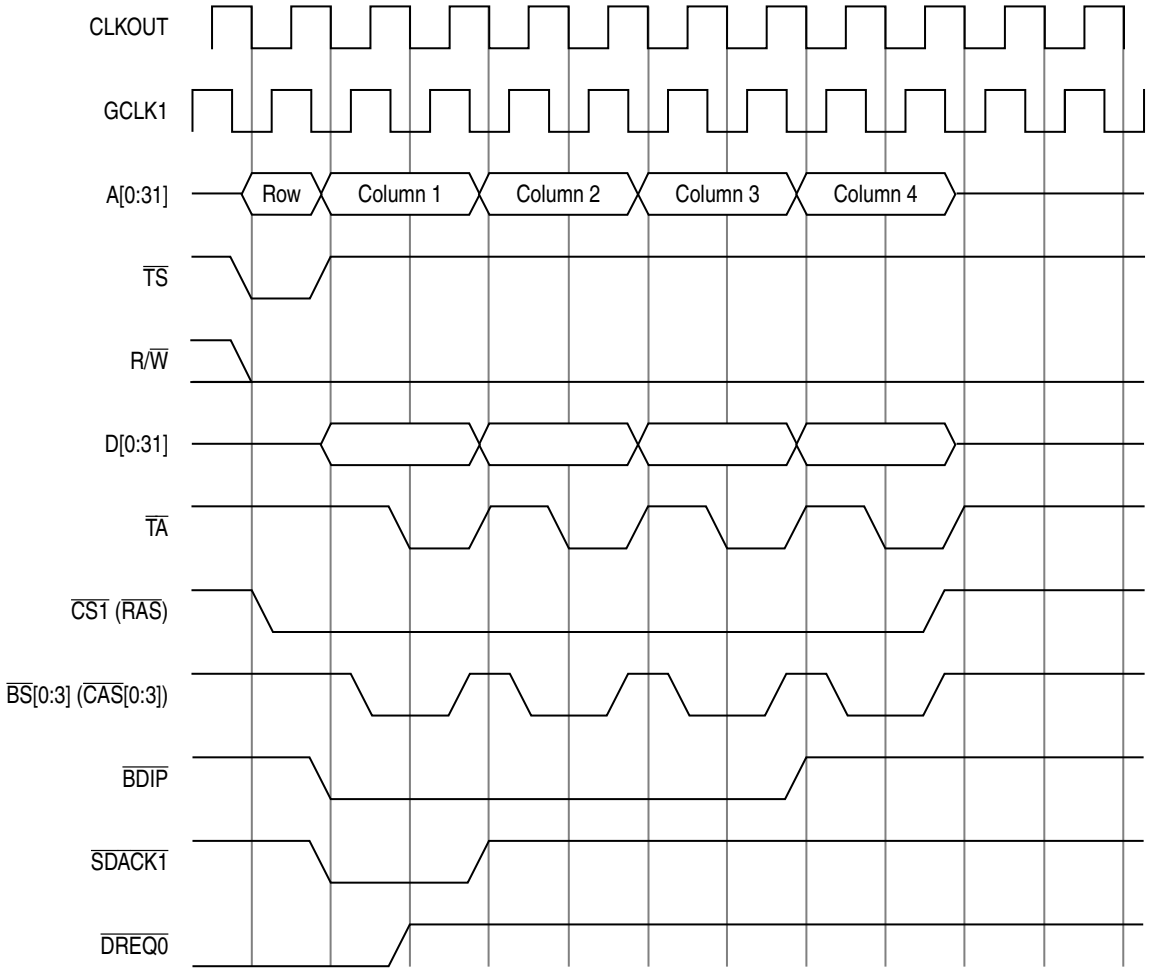


Figure 19-15. Single-Address IDMA1 Burst Timing (Single-Buffer Mode)

19.3.10 External Recognition of an IDMA Transfer

The following are ways to externally determine if IDMA is executing a bus cycle:

- Monitor the AT signals of the SDMA channels for the user-defined function code. (AT0 is always high for a DMA access.)
- Monitor SDACK, which shows accesses to the peripheral. SDACK activates on either the source or destination bus cycles, depending on DCMR[S/D]. Note that if Ethernet is running, this method does not work since SCC in Ethernet mode also toggle SDACK for SDMA transfers.

19.3.11 Interrupts During an IDMA Bus Transfer

The MPC855T supports a synchronous bus structure with provisions allowing a bus master to detect and respond to errors during a bus cycle. An IDMA channel recognizes the same bus interrupt sources that the core recognizes—reset and transfer error acknowledge ($\overline{\text{TEA}}$).

- Reset—On an external reset, an IDMA immediately aborts channel operation, returns to the idle state, and clears the IDSR. If a bus cycle is in progress, the cycle is terminated, the control and address/data pins are three-stated, and the bus ownership is released. Program control passes to the handler at the system reset interrupt vector (0x00100).
- Transfer error acknowledge ($\overline{\text{TEA}}$)—When a fatal error occurs during an IDMA bus cycle, $\overline{\text{TEA}}$ is used to abort the cycle and systematically terminate the channel's operation. The IDMA terminates the current bus cycle, flags an error in SDSR and interrupts the core if not masked by SDMR. The IDMA waits for the CPM to reset before starting any new bus cycles. Note that data read from the source into internal storage is lost. Program control passes to the handler at the machine check interrupt vector (0x00200).

The machine check and system reset interrupts are described in Chapter 6, "Exceptions."

Note that the source or destination device 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 $\overline{\text{SDACK}}$, which is used to control the transfer to or from the device.

Chapter 20
Serial Interface

The physical interface to the SCC and SMCs is implemented in the serial interface (SI). The SI allows each individual SCC and SMC to be connected externally either through a time-division multiplexed (TDM) interface or through dedicated pins in a non-multiplexed serial interface (NMSI).

When the SCC or SMC is configured to use the NMSI, the SI provides flexible clocking from a bank of clocks, including external clock pins and internal baud rate generators (BRGs). See Section 20.3, “NMSI Configuration,” and Section 20.4, “Baud Rate Generators (BRGs).”

Connecting the SCC and SMCs to the multiplexed (TDMa) interface is accomplished through a set of TDMa pins and a time-slot assigner (TSA). The user programs the TSA to route data from the TDM data stream to and from the SCC and SMCs. The TSA also provides external strobe signals (L1ST1–4), which can be used to enable external devices, such as codecs, to insert or take data from the TDM data stream. An external framer (providing receive and transmit data, clocks, and synchronization signals) is required to interface to the TDM channel. Common examples of TDM channels are T1 lines in the U.S. and Japan and CEPT (E1) lines in Europe.

If the TSA is not required for routing data to and from the SCC and SMCs, it can still be used to generate complex waveforms on its four strobe output pins (L1ST[1–4). For example, the user can program the TSA to implement stepper motor control signals of variable duty cycle and period.

Figure 20-1 shows the SI block diagram.

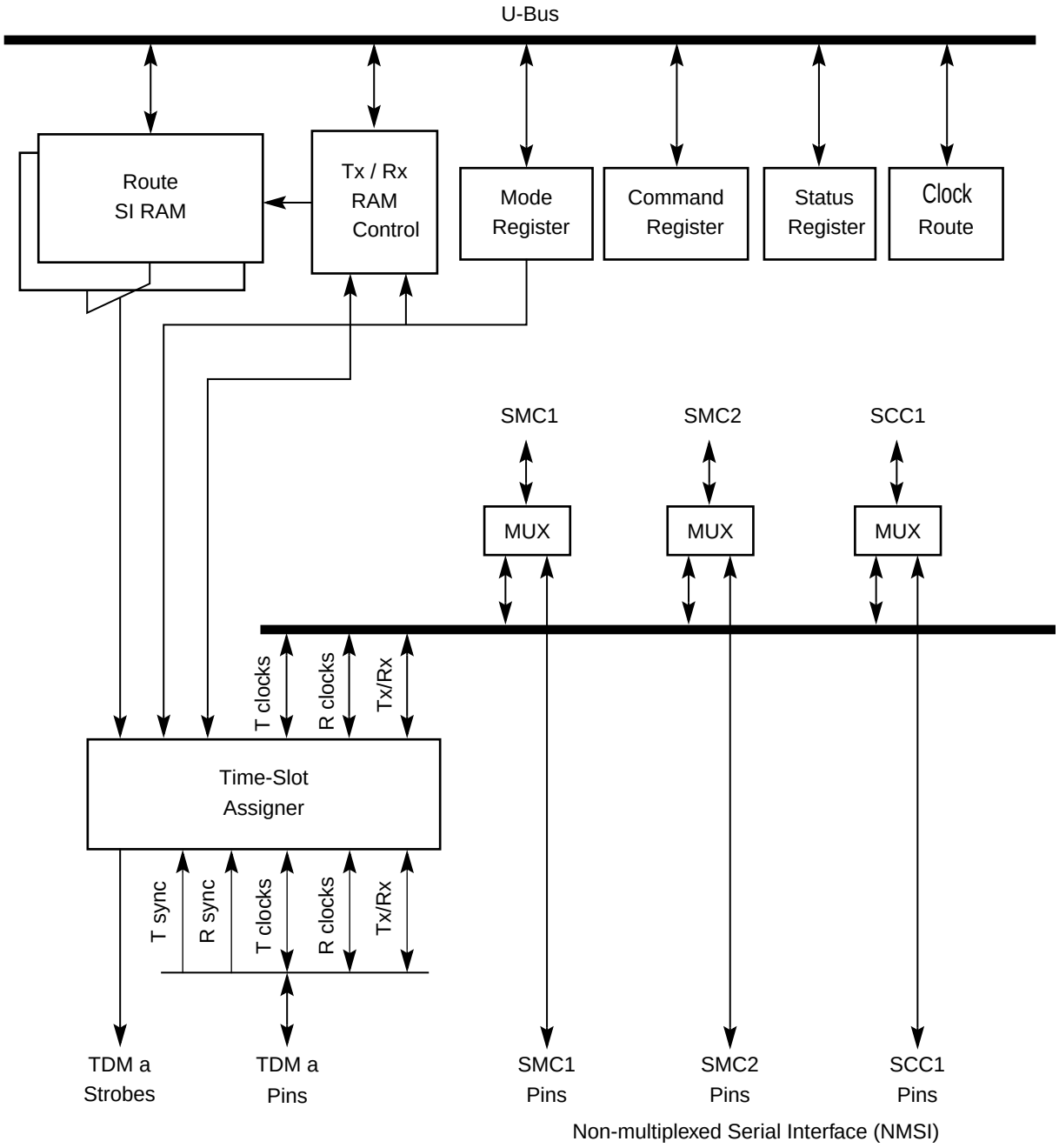


Figure 20-1. MPC855T SI Block Diagram

20.1 SI Features

The TSA's main features are as follows:

- Ability to connect the TDM channel as follows:
  - T1 or CEPT line
  - Pulse code modulation highway (PCM)

  - User-defined interfaces
- Independent Tx and Rx routing paths programmed in the SI RAM
- Independent Tx and Rx frame syncs
- Independent Tx and Rx 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
- Four programmable strobe outputs and two (1x) clock output pins
- Bit or byte resolution in routing, masking, and strobe selection
- Supports frames up to 8,192 bits long
- Internal routing and strobe selection can be programmed dynamically.
- Supports automatic echo and loopback modes for the TDM channel

The NMSI is discussed in Section 20.3, “NMSI Configuration.” Its main features are as follows:

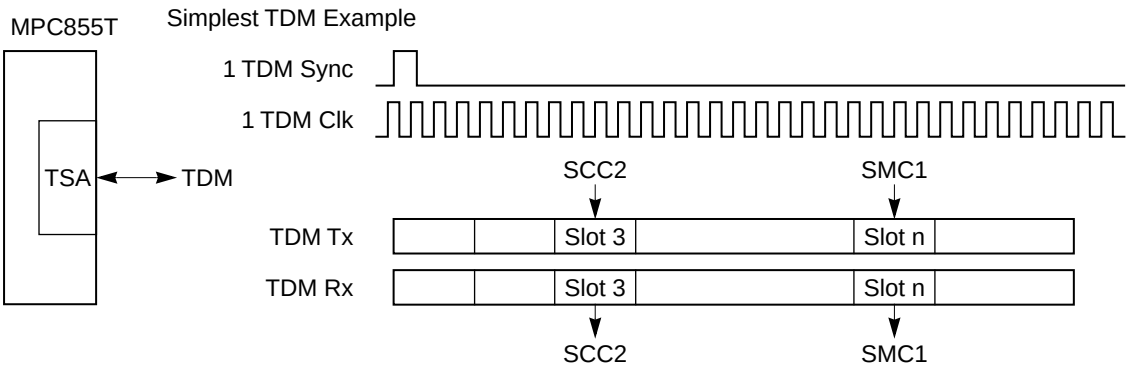
- Each SCC and SMC can be programmed independently to work with its own set of non-multiplexed signals
- Each SCC can have its own set of modem control signals
- Each SMC can have its own set of four signals
- Each SCC and SMC can derive clocks externally from a bank of eight clock signals or a bank of four baud-rate generators

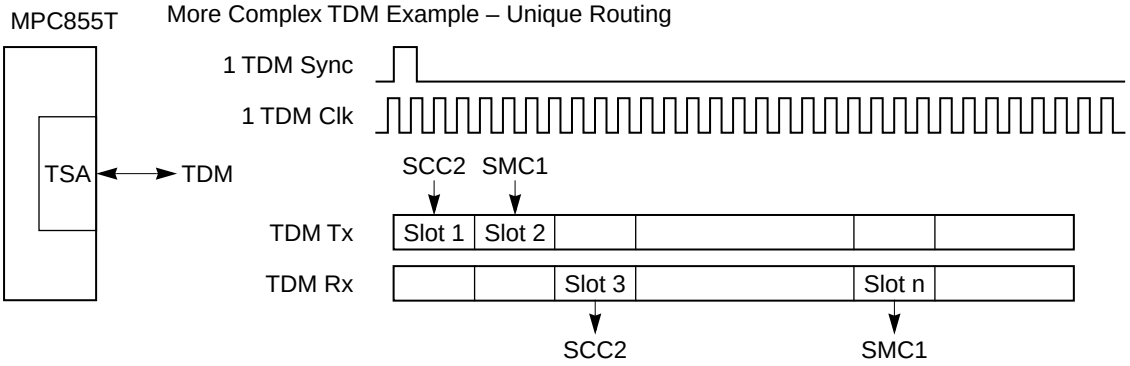
20.2 The Time-Slot Assigner (TSA)

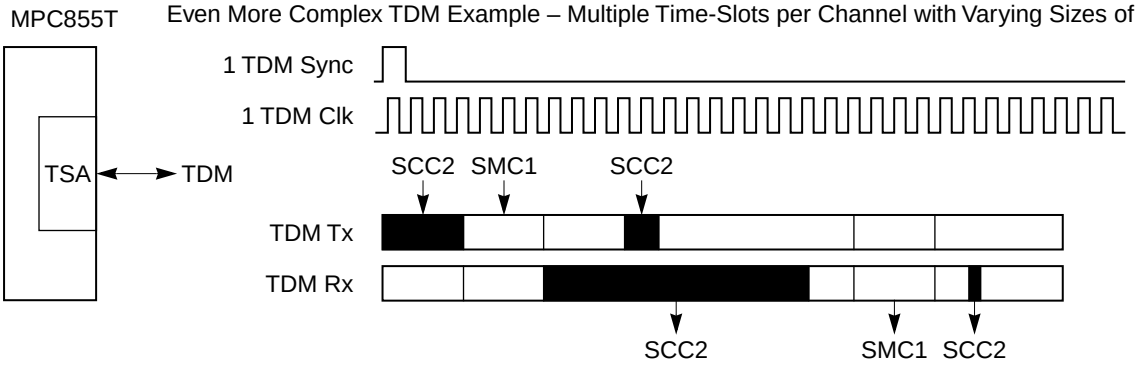
The time-slot assigner (TSA) implements both internal route selection and time-division multiplexing (TDM) for multiplexed serial channels. The TSA supports the serial bus rate and format for most standard TDM buses, including the T1 and CEPT highways, and pulse code modulation (PCM) highway.

TSA programming is independent of the protocol used by the SCC or SMC. The SCC and SMCs can be programmed for any synchronous protocol without affecting TSA programming. The TSA simply routes programmed portions of the received data frame from the TDM pins to the target SCC and SMCs, while the target SCC or SMC handles the received data in the actual protocol.

In its simplest mode, the TSA identifies both Rx and Tx frames using one sync pulse and a single clock signal provided by the user externally. This mode can be enhanced to allow independent routing of Tx and Rx data on the TDM channel. The user defines the length of a time slot, which need not be limited to 8 bits or even to a single contiguous position within the frame. For more flexibility, the user can also provide separate Rx and Tx syncs as well as independent clocks. Figure 20-2 shows example TSA configurations ranging from the simplest to the most complex.







NOTE: The two shaded areas of SCC2 Rx are received as one high-speed data stream by the S and stored together in the same data buffers.

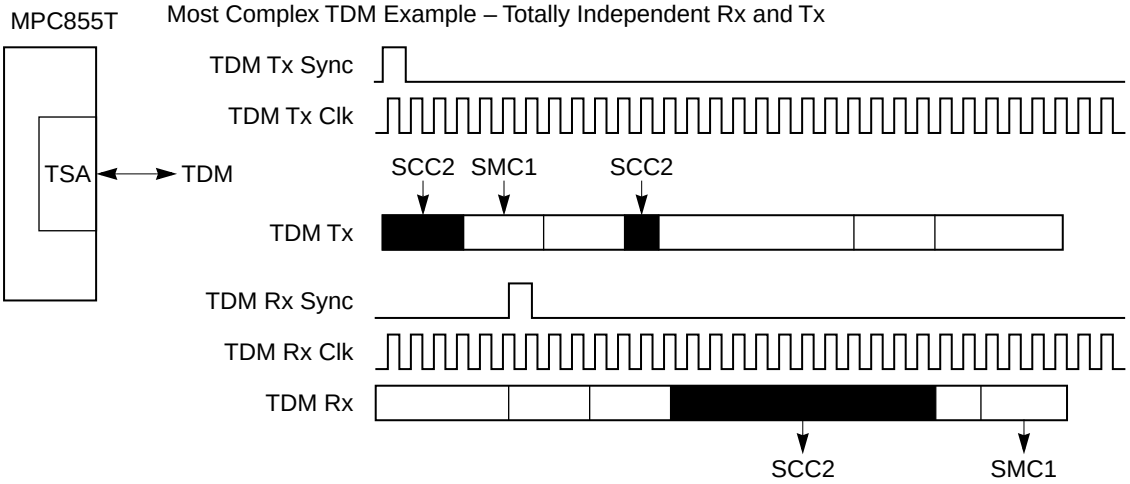


Figure 20-2. Various Configurations of a TDM Channel

The TSA can support two, independent, half-duplex TDM sources, one receiving and one transmitting, using two sync inputs and two input clocks. In addition to channel programming, the TSA supports up to eight strobe outputs that may be asserted on a bit basis or a byte basis. These strobes are completely independent from the channel routing used by the SCC and 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. (Note that open-drain programming on the TXDx pins to support a multi-transmitter architecture is programmed in the parallel I/O block.) These strobes can also generate output waveforms for such applications as stepper motor control.

The TSA routing is programmed in a 512-byte, core-accessible SI RAM located in the internal register section of the MPC855Tseparate from the dual-port RAM. The SI RAM contains a total of 128 32-bit entries: the first 64 entries are for programming receive routing, and the second 64 are for transmit routing. The entries define the number of bits or bytes to be routed to and from the SCC or SMCs and also control external strobes.

The amount of SI RAM available for time-slot programming depends on the configuration of the SIGMR; see Section 20.2.4.1, “SI Global Mode Register (SIGMR).” Using all 64 entries of the Rx or Tx SI RAM, TDMa can support a maximum frame length of 8,192 bits. Enabling on-the-fly changes divides the SI RAM to allow for routing workspace. See Section 20.2.3, “SI RAM.”

The SI supports two testing modes—echo and loopback. The echo mode provides a return signal from the physical interface by retransmitting the signal it receives. The physical interface echo mode differs from the individual SCC echo mode in that it operates on the entire TDM signal rather than on an individual SCC channel. Loopback mode causes the physical interface to receive the same signal it is sending. Checking both the entire SI and the internal channel routes, the SI loopback mode does more than the individual SCC loopback. Programming echo and loopback modes are programmed in SIMODE[SDMa]; see Section 20.2.4.2, “SI Mode Register (SIMODE).” Loopback mode can also be programmed on a time-slot basis in an individual SI RAM entry; see Section 20.2.3.5, “Programming the SI RAM.” Note that loopback operation requires that the receive and transmit sections of the TDM use common clock and synchronization signals.

The maximum external serial clock that may be an input to the TSA is SYNCCLK/2.5. If the SCC or SMC is operating with the NMSI, the serial clock rate may be slightly faster at a value not to exceed SYNCCLK/2.25.

Note that a sync pulse received during TSA frame routing is ignored. However, when programmed for a one-clock delay between the sync and start-of-frame pulses, the TSA can accept the last bit of a frame overlapping the sync pulse of the next frame.

20.2.1 TSA Signals

The TSA signals for TDMa are shown in Table 20-1.

Table 20-1. TSA Signals

Signal	Description
L1RSYNCa/L1TSYNCa	Receive/transmit synchronization signals. Input to the MPC855T.
L1RCLKa/L1TCLKa	Receive/transmit clocks. Input to the MPC855T.
L1RXDa	Receive data. Input to the MPC855T.
L1TXDa	Transmit data. Open-drain output of the MPC855T.

Note that if the receive and transmit clocks and the synchronization signals are common, L1TSYNCa and L1TCLKa are not needed.

20.2.2 Enabling Connections to the TSA

Each SCC and SMC can be independently enabled to connect to the TSA. The SCC is connected to the TSA by programming the SI clock route register SICR[SC*x*]; see Section 20.2.4.3, “SI Clock Route Register (SICR).” The SMCs are connected to the TSA by setting the mode register SIMODE[SMC*x*]; see Section 20.2.4.2, “SI Mode Register (SIMODE).” The general mode register SIGMR[ENa] must also be set to enable TDMa; see Section 20.2.4.1, “SI Global Mode Register (SIGMR).” Once the connections are made, the exact routing is determined in the SI RAM. See Figure 20-3.

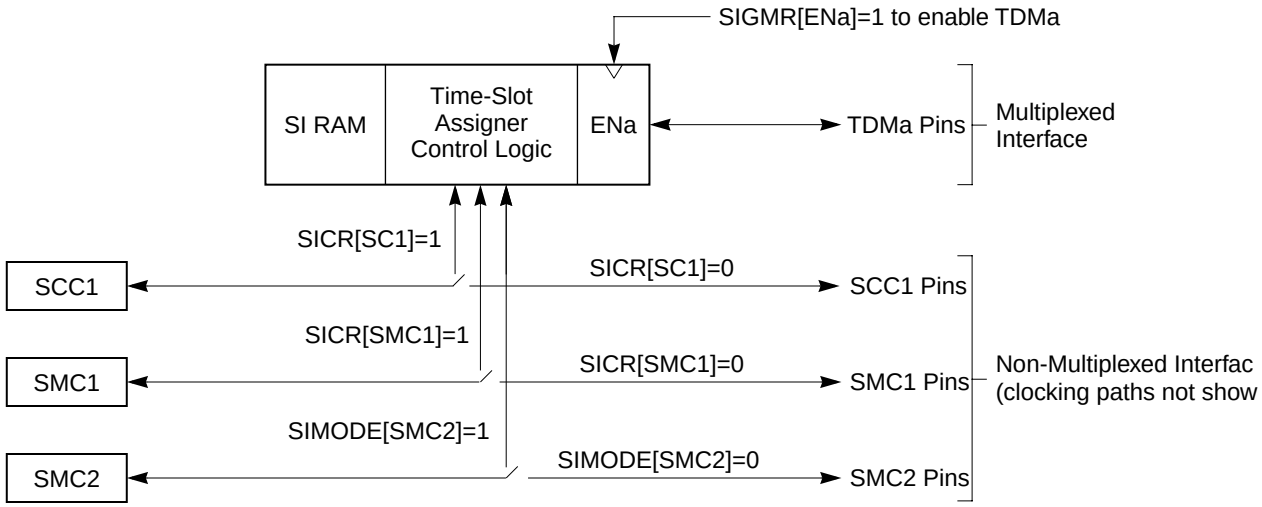


Figure 20-3. Enabling Connections through the SI

20.2.3 SI RAM

The 512-byte SI RAM contains the SCC and SMC routing information for the TDM channel. The SI RAM totals 128 32-bit entries—64 entries each for receive and transmit

routing. Representing one time slot, an entry controls from 1 to 16 bits/bytes and up to four strobe pins (all active high).

The TDM channel options with their corresponding SI RAM partitioning follow:

- A single TDM channel with static routing—SI RAM is divided into Rx and Tx parts.
- A single TDM channel with dynamic routing—Rx and Tx RAMs are halved.

Note that the SI RAM is uninitialized after power-on—the core should program them before enabling the TDM channel.

20.2.3.1 Disabling and Reenabling the TSA

The following steps must be taken any time the TSA is disabled. These steps also apply to changing the SI routing when the TSA is configured for static frames.

1. SCC and SMC connections to the TSA must be disabled.
2. The SI configuration can be modified.
3. SCC and SMC connections to the TSA must be reenabled.

20.2.3.2 TDMa Channel with Static Frames

In an SI configuration using one multiplexed channel with static frames, shown in Figure 20-4, there are 64 entries in the SI RAM for Rx data/strobe routing and 64 entries for Tx data/strobe routing.

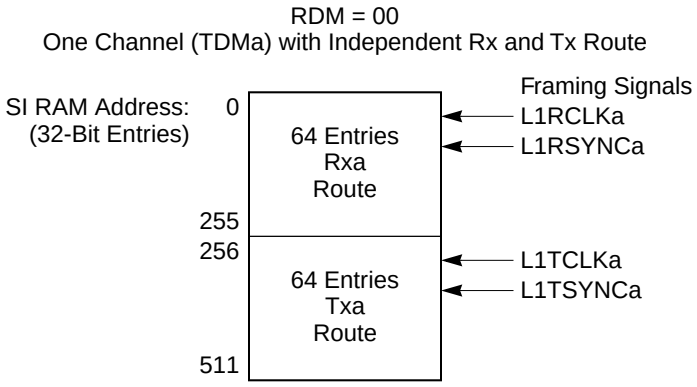


Figure 20-4. SI RAM Partitioning Using TDMa with Static Frames

20.2.3.3 SI RAM Dynamic Changes

The routing of a TDM channel can be changed while the SCC and SMCs remain connected to the TSA. Enabling dynamic changes divides the SI RAM into current-route and work-space shadow areas.

Once the current-route RAM is programmed, the TDM channel can be enabled and SI operation begun. New routing information can then be programmed into the shadow RAM.

Setting the channel's change-shadow-RAM bits, SICMR[CSRRa, CSRTa], in the SI command register tells the SI to activate the shadow RAM (deactivating the current-route RAM) when the next frame sync arrives. The SI signals the user by clearing SICMR[CSRRa, CSRTa] when the swap takes effect. These steps can be repeated with the former current-route RAM always becoming the new shadow RAM and vice versa.

When using one channel (TDMa) with dynamic changes, as in Figure 20-5, the initial current-route RAM byte addresses are as follows.

- 0–127 RXa route
- 256–383 TXa route

The shadow RAMs are at addresses:

- 128–255 RXa route
- 384–511 TXa route

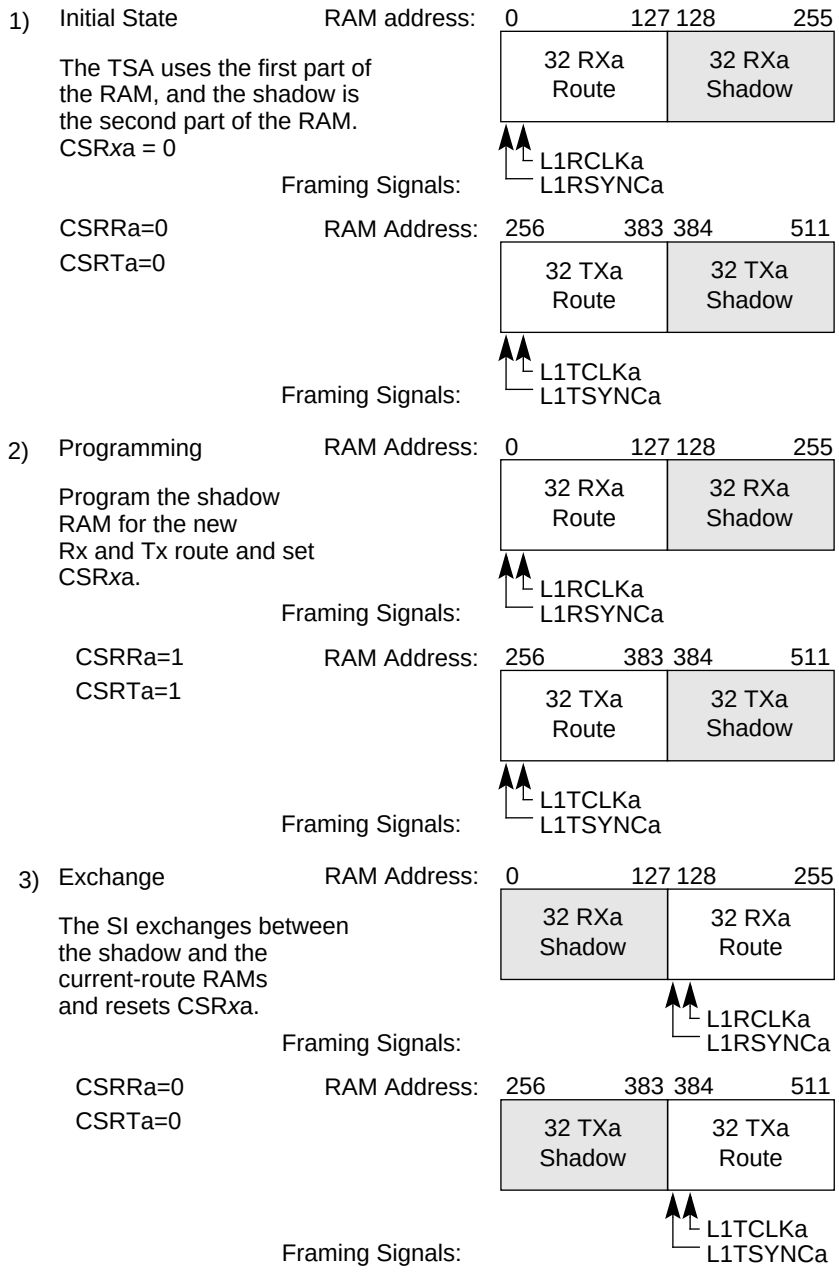


Figure 20-5. SI RAM Dynamic Changes with TDMa

The entire SI RAM is always readable, but only the shadow RAM is safe to write. The SI status register (SISTR) can be read to determine which part of the RAM is the current-route RAM. The SI RAM pointer (SIRP) register can be used to determine which SI RAM entry is active. In addition, by externally connecting a strobe to an interrupt signal, an individual SI RAM entry can generate an interrupt.

20.2.3.4 TDMa Channel with Dynamic Frames

In an SI configuration using the one TDM channel with dynamic frames, TDMa has 32 entries apiece for Tx and Rx data/strobe routing, as shown in Figure 20-6. One RAM partition is the current-route RAM; the other is shadow RAM that can be safely reprogrammed. After programming the shadow RAM, set the CSRa bit of the channel in the SI command register (SICMR). When the next frame sync arrives, the SI swaps the current-route RAM with the shadow RAM.

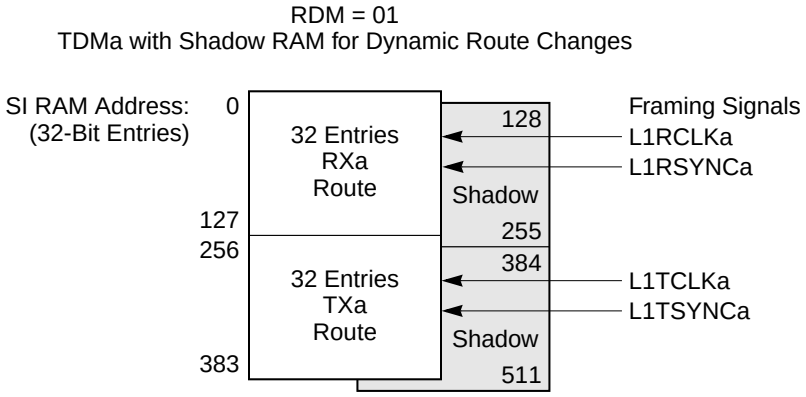


Figure 20-6. SI RAM Partitioning Using TDMa with Dynamic Frames

20.2.3.5 Programming the SI RAM

Each SI RAM entry determines the routing of the serial bits and the state of strobe outputs for one time slot. Figure 20-7 shows the format of an SI RAM entry.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	LOOP	SWTR	SSEL4	SSEL3	SSEL2	SSEL1	—	CSEL			CNT				BYT	LST
Reset	0															
R/W	R/W															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—															
Reset	0															
R/W	R/W															

Figure 20-7. SIRAM Entry

Table 20-2 describes SI RAM entry fields.

Table 20-2. SIRAM Field Descriptions

Bits	Name	Description
0	LOOP	Loop back on this time slot. 0 Normal mode (no loopback). 1 Loopback for this time slot.
1	SWTR	Switch transmit and receive. Valid only in Rx route RAM; ignored in the Tx route RAM. Affects operation of both L1RXDa and L1TXDa. See Figure 20-8 and the accompanying text. 0 Normal operation of L1TXDa and L1RXDa. 1 Data is sent on L1RXDa and received from L1TXDa for the duration of this entry. Note that erratic results may occur if the Tx and Rx sections of the TDM do not use a common clock source.
2–5	SSEL*n*	Strobe select 1–4. The four strobes, L1ST[1–4], can be assigned to the Rx or the Tx RAM and asserted/negated in sync with the corresponding L1RCLKa or L1TCLKa. Using active-high logic, each SSEL*n* will be the value of the corresponding strobe during this time slot. Multiple strobes can be asserted simultaneously. A strobe can be configured to remain asserted for multiple, consecutive SI RAM entries; however, if a strobe is asserted on the last entry in the table, the strobe is negated after the last entry finishes processing. Note: The corresponding parallel I/O pins (either port B or C) must be configured for strobe operation; see Chapter 33, “Parallel I/O Ports.”
6	—	Reserved, should be cleared.
7–9	CSEL	Channel select. Indicates which channel the time slot is routed to. 000 This time slot is not used. Tx data signal is three-stated; Rx data signal is ignored. 001 SCC1 010 Reserved. Do not use. 011 Reserved. Do not use. 100 Reserved. Do not use. 101 SMC1 110 SMC2 111 This time slot is not used. Also used in SCIT mode to indicate the D channel grant bit.
10–13	CNT	Count. The number of bits/bytes (the unit is determined by BYT) that the routing and strobe select of this entry controls. If CNT = 0b0000, 1 bit/byte is routed; if CNT = 0b1111, 16 bits/bytes are routed.
14	BYT	Byte resolution. 0 Bit resolution. CNT indicates the number of bits in this entry. 1 Byte resolution. CNT indicates the number of bytes in this entry.
15	LST	Last entry in a TDM channel’s Rx or Tx SI RAM. LST must be set in the last entry even if all the entries are used. 0 Not the last entry in this TDM channel’s Rx or Tx SI RAM. 1 Last entry. After this entry, the SI waits for SYNC to start the next frame.
16–31	—	Reserved

For applications needing to receive data from a Tx signal and send data on an Rx signal, set SIRAM*n*[SWTR]. For example, stations A and B in Figure 20-8 use different time slots on one TDM channel. Even though they share Rx and Tx data lines, stations A and B can communicate using the SWTR option.

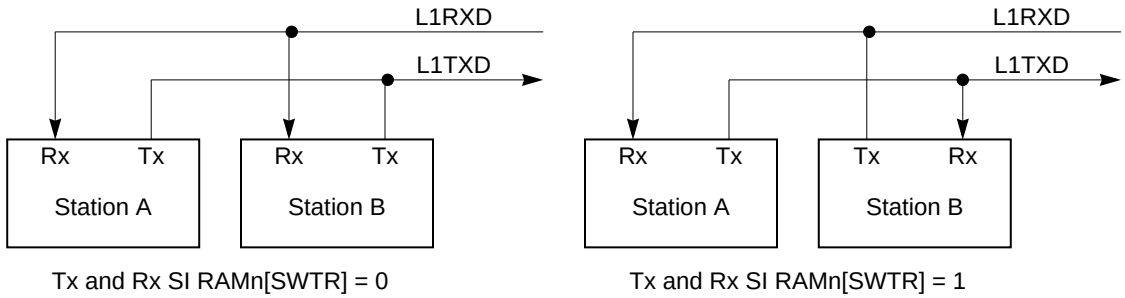


Figure 20-8. Example Using SI RAM*n*[SWTR]

The SWTR option allows station B to listen to transmissions from and send data to station A. By setting SWTR in its Rx route RAM entry, station B receives data from L1TXD and, if the time slot's Tx route RAM entry allows, sends data on L1RXD. To prevent sending on L1RXD while listening to station A, clear the CSEL bits in the corresponding Tx route RAM entries. Conversely, to prevent receiving on L1TXD while sending on L1RXD, clear the CSEL bits in corresponding Rx SI RAM entries. Note that using the SWTR option may cause data collisions with other stations unless an empty (quiet) time slot is used.

20.2.4 The SI Registers

The following sections describe the SI registers.

20.2.4.1 SI Global Mode Register (SIGMR)

The SI global mode register (SIGMR), shown in Figure 20-9, defines the SI RAM division modes and enables the TDM channel.

Bit	0	1	2	3	4	5	6	7
Field	—				—	ENa	RDM	
Reset	0							
R/W	R/W							
Addr	0xAE4							

Figure 20-9. SI Global Mode Register (SIGMR)

This register is affected by $\overline{\text{HRESET}}$ but is not affected by $\overline{\text{SRESET}}$. Table 20-3 describes the SIGMR fields.

Table 20-3. SIGMR Field Descriptions

Bits	Name	Description
0–4	—	Reserved, should be cleared.
5	ENa	Enable TDMa. 0 TDMa is disabled. SI RAM and TDM routing are in a state of reset; all other SI functions still operate. 1 TDMa is enabled.
6–7	RDM	RAM division mode. Defines the SI RAM partitioning based on whether dynamic changes are needed. 00 Static TDMa with 64 entries apiece for Rx and Tx routing. 01 Dynamic TDMa with 32 entries apiece for current-route and shadow Rx routing and 32 apiece for current-route and shadow Tx routing. 1x Reserved.

Note that after setting SIGMR[ENa], data from the transmit buffers does not immediately appear at the TDM transmit pin with the first frame because the SCC requires start-up clocking at initialization. Expect a number of bytes of idle (typically 10–15) depending on the size of the frame and number of time slots routed to the SCC.

20.2.4.2 SI Mode Register (SIMODE)

The SI mode register (SIMODE), shown in Figure 20-10, defines the SI operation modes for the TDM channel and SMCs.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	SMC2	SMC2CS			—											
Reset	0															
R/W	R/W															
Addr	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															
R/W	R/W															
Addr	0xAE2															

Figure 20-10. SI Mode Register (SIMODE)

This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Table 20-4 describes the SIMODE fields.

Table 20-4. SIMODE Field Descriptions

Bits	Name	Description
0, 16	SMC*x*	SMC*x* connection 0 NMSI mode. The clock source is determined by SMC*x*CS and the data comes from a dedicated pin (SMTXD1 and SMRXD1 for SMC1 or SMTXD2 and SMRXD2 for SMC2) in NMSI mode. 1 SMC*x* is connected to the multiplexed SI (TDM channel).
1–3, 17–19	SMC*x*CS	SMC*x* clock source (NMSI mode). SMC*x* can take its Tx and Rx clocks from a baud rate generator or one of four pins from the bank of clocks. However, Tx and Rx clocks must be common when connected to the NMSI. 000 BRG1 001 BRG2 010 BRG3 011 BRG4 100 CLK1 101 CLK2 110 CLK3 111 CLK4
20–21	SDMa	SI diagnostic mode for TDMa. In modes 01,10, and 11, Rx and Tx clocks should be common. 00 Normal operation. 01 Automatic echo. The TDM transmitter automatically resends its Rx data bit-by-bit. The Rx section operates normally, but the Tx section can only resend Rx data. L1GRa is ignored. 10 Internal loopback. TDM transmitter output is connected internally to the TDM receiver input—L1TXDa is connected to L1RXDa. The receiver and transmitter operate normally, but data on L1RXDa is ignored. Data appears on L1TXDa, $\overline{\text{L1RQa}}$ is asserted normally, and L1GRa is ignored. 11 Loopback control. TDM transmitter output is connected internally to the TDM receiver input—L1TXDa is connected to L1RXDa. Transmitter output L1TXDa and $\overline{\text{L1RQa}}$ are inactive. Provides loopback testing of the entire TDM without affecting the external serial lines.
22–23	RFSDa	Receive frame sync delay for TDMa. Indicates the delay between the Rx sync and the first bit of the Rx frame. Even if CRTa is set, RFSDa does not control the Tx frame delay. 00 No bit delay. The first bit of the frame is received on the same clock as sync. 01 1-bit delay. 10 2-bit delay. 11 3-bit delay. See the examples in Figure 20-11 and Figure 20-12.
24	DSCa	Double speed clock for TDMa—for TDM interfaces that define the input clock to be twice as fast as the data rate. 0 Channel clock (L1RCLKa and/or L1TCLKa) is equal to the data clock. Use for most TDM formats. 1 Channel clock rate is twice the data rate.
25	CRTa	Common receive and transmit pins for TDMa. 0 Separate pins. The receive section of the TDM uses L1RCLKa and L1RSYNCa for framing; the transmit section uses L1TCLKa and L1TSYNCa for framing. 1 Common pins. Both the transmit and receive section use L1RCLKa as the clock pin of the channel and L1RSYNCa as the sync pin. Useful when the transmit and receive section of a given TDM share clock and sync signals. L1TCLKa and L1TSYNCa can be used for their alternate functions.

Table 20-4. SIMODE Field Descriptions (continued)

Bits	Name	Description
26	STZa	Set L1TXDa to zero for TDMa. 0 Normal operation. 1 L1TXDa is cleared until serial clocks are available.
27	CEa	Clock edge for TDMa. When DSCa = 0: 0 Data is sent on the rising clock edge and received on the falling edge. 1 Data is sent on the falling edge of the clock and received on the rising edge. When DSCa = 1: 0 Data is sent on the rising clock edge and received on the rising edge. 1 Data is sent on the falling edge of the clock and received on the falling edge.
28	FEa	Frame sync edge for TDMa. Indicates when L1RSYNCa and L1TSYNCa pulses are sampled with the falling/rising edge of the channel clock. 0 Falling edge. 1 Rising edge.
29	GM	Grant mode for TDMa. 0 GCI/SCIT mode. The GCI/SCIT D channel grant mechanism for transmission is supported internally. The grant is signalled by one bit of the Rx frame and is marked by setting SIRAM[CSEL] to 111 to assert an internal strobe. 1 IDL mode. A grant mechanism is supported if the corresponding SICR[GR*n*] are set. The grant is a sample of the L1GRa signal while L1TSYNCa is asserted. This grant mechanism implies the IDL access controls transmission on the D channel.
30–31	TFSDa	Transmit frame sync delay for TDMa. Determines the delay between the Tx sync and the first bit of the Tx frame. If CRTa is set, the Rx sync is used as the common sync, and the TFSDa bits refer to this common sync. 00 No bit delay. The first bit of the frame is sent on the same clock as the sync. 01 1-bit delay. 10 2-bit delay. 11 3-bit delay.

The following series of figures show timing examples. Figure 20-11 and Figure 20-12 show the effects of changing the delay from frame sync to data valid.

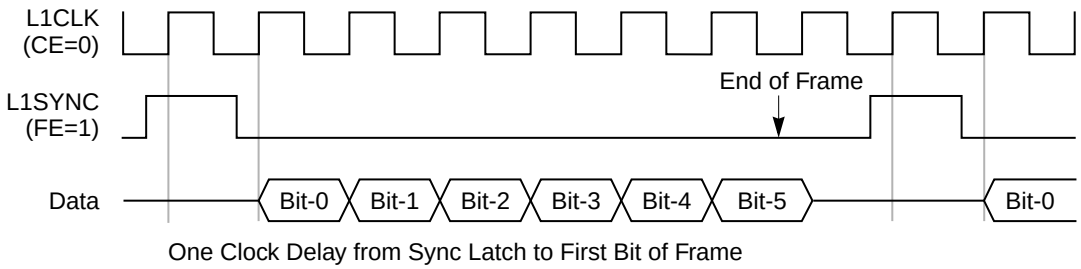


Figure 20-11. One Clock Delay from Sync to Data (*x*FSD = 01)

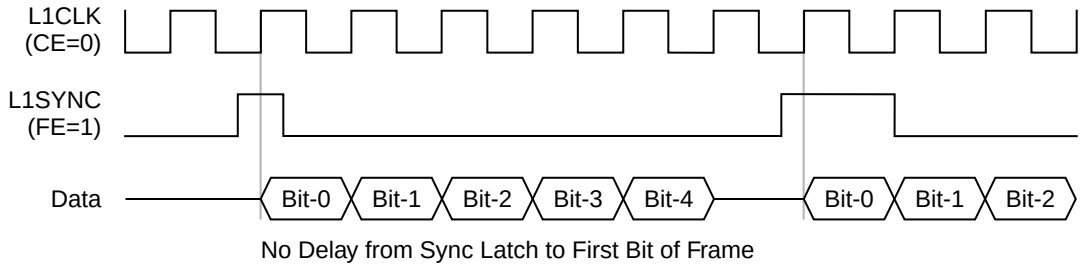


Figure 20-12. No Delay from Sync to Data (*x*FSD = 00)

Figure 20-13 and Figure 20-14 show example timings while varying SIMODE[CE] with a constant frame sync delay of one bit.

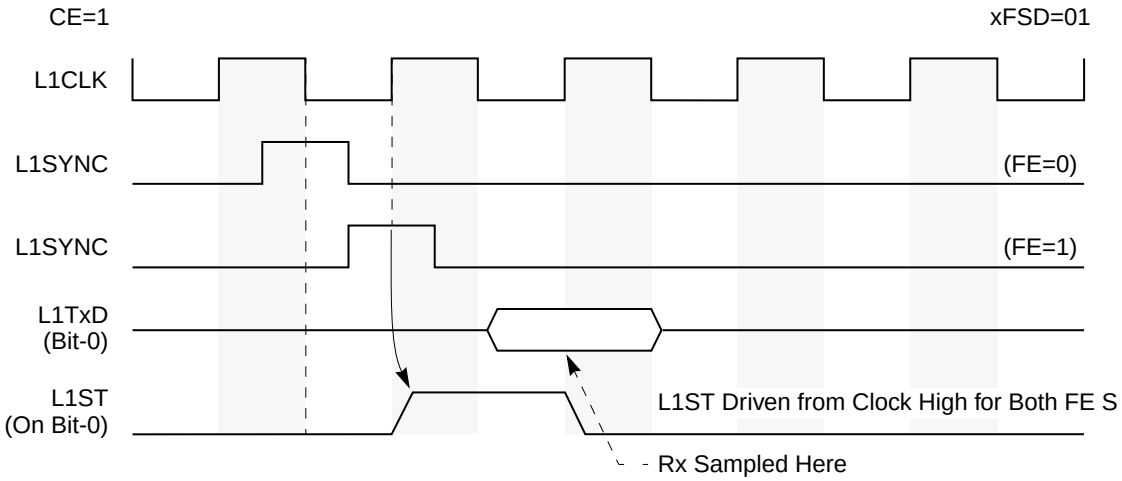


Figure 20-13. Falling Edge (FE) Effect When CE = 1 and *x*FSD = 01

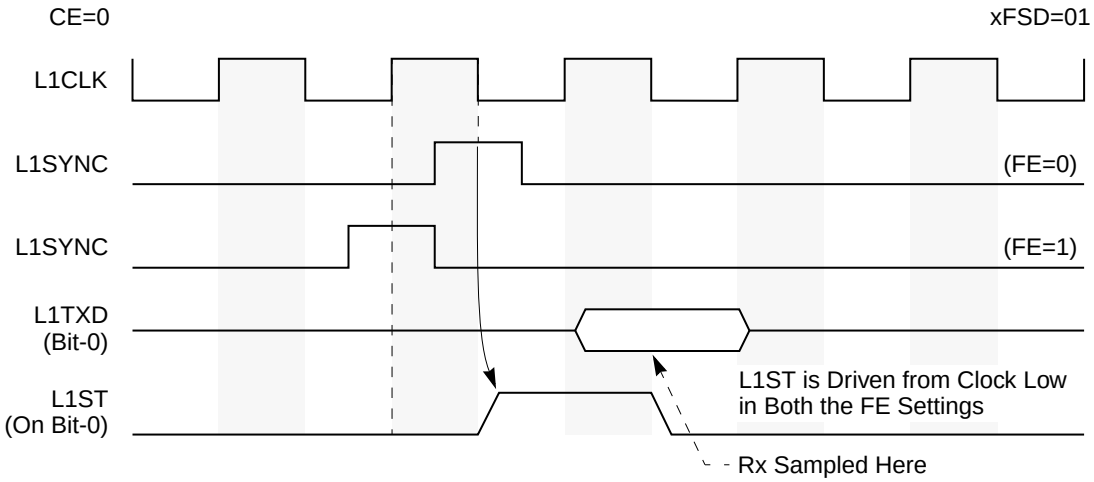


Figure 20-14. Falling Edge (FE) Effect When CE = 0 and *x*FSD = 01

Figure 20-15 shows SIMODE[FE] behavior with SIMODE[CE] set and no frame sync delay.

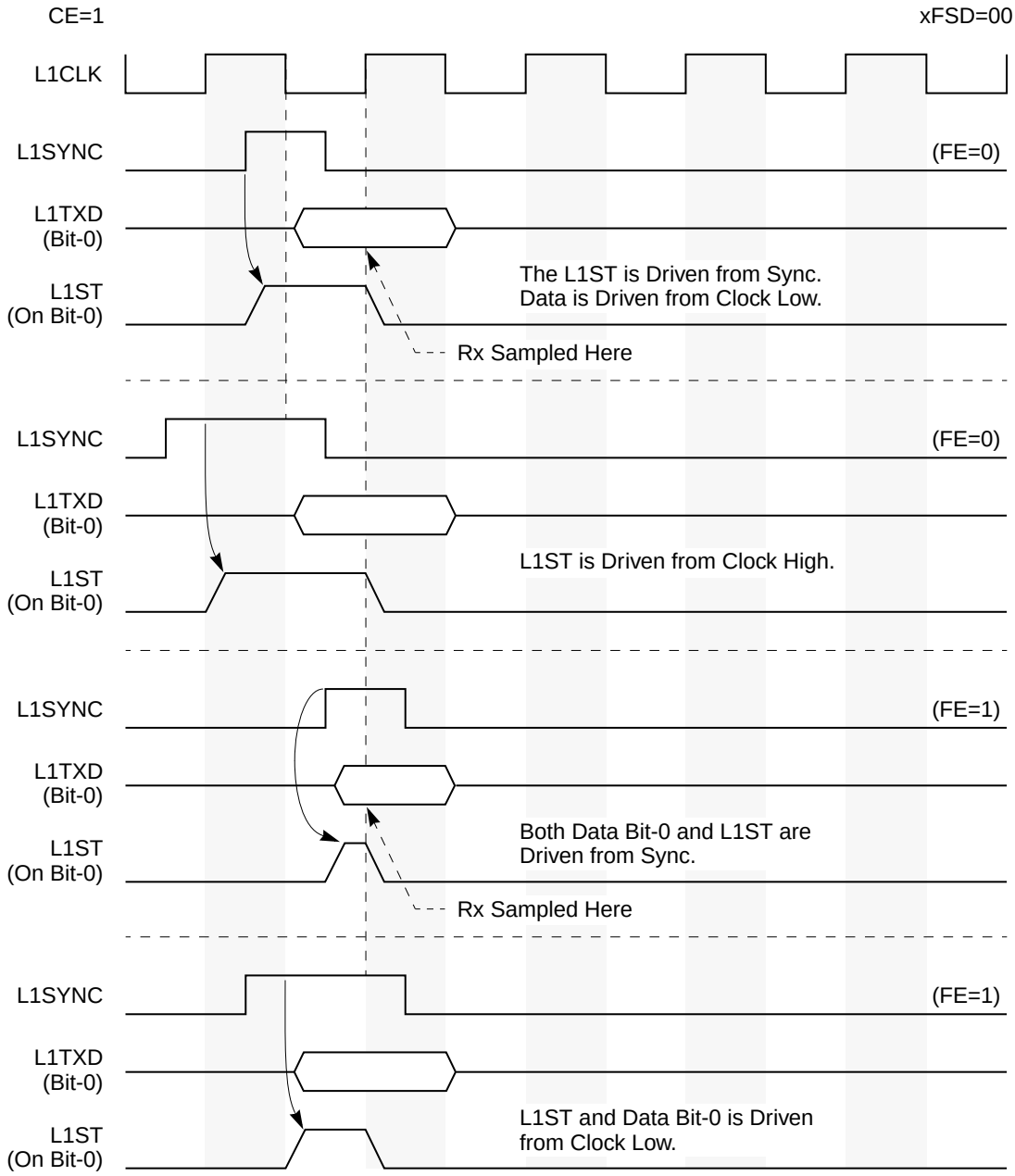


Figure 20-15. Falling Edge (FE) Effect When CE = 1 and *x*FSD = 00

Figure 20-16 shows SIMODE[FE] behavior when SIMODE[CE] and SIMODE[*x*FSD] are zero.

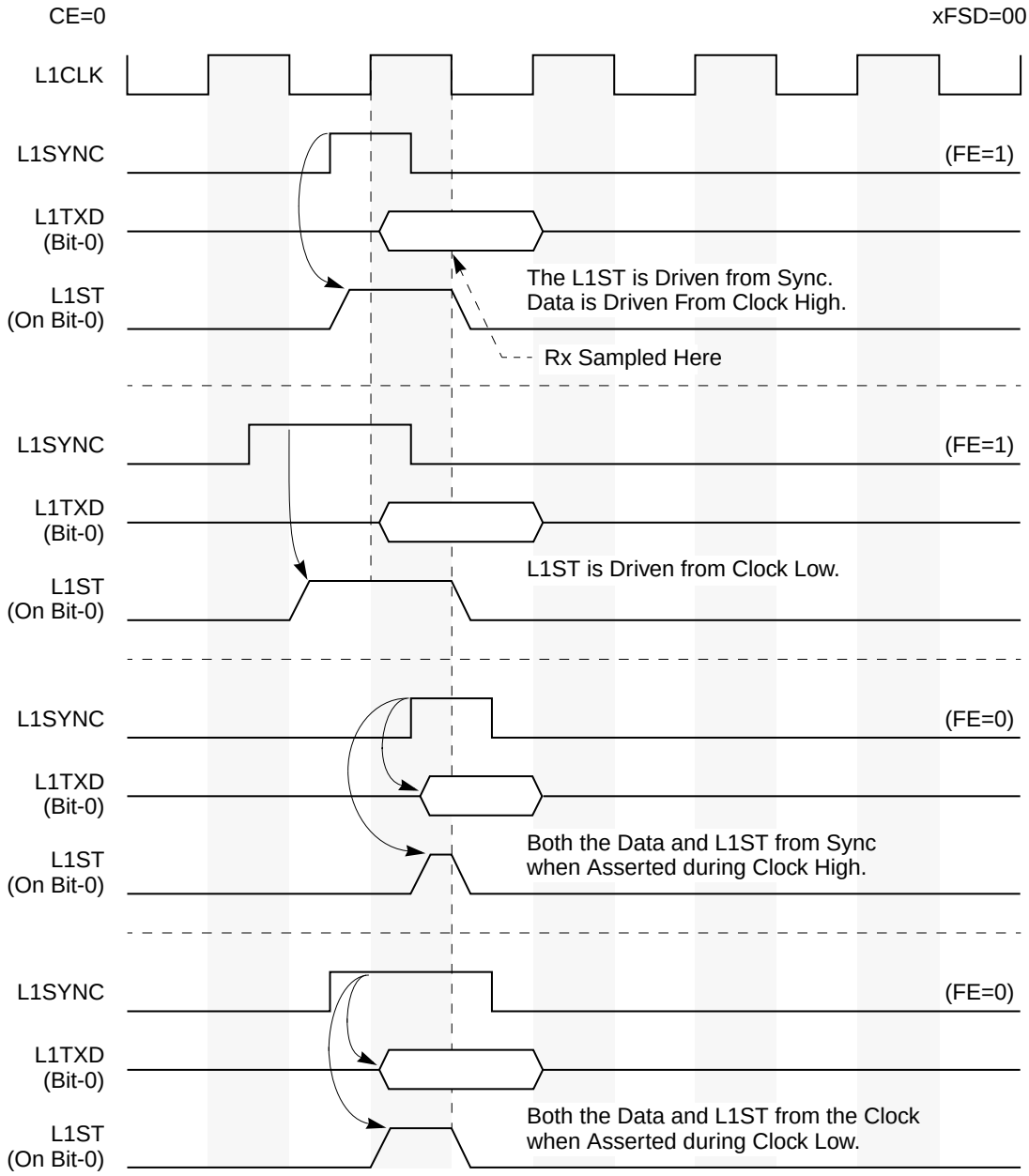


Figure 20-16. Falling Edge (FE) Effect When CE = 0 and *x*FSD = 00

20.2.4.3 SI Clock Route Register (SICR)

The SI clock route register (SICR), shown in Figure 20-17, selects the SCC clock source from one of four baud rate generators or an input from the bank of clock pins. The SICR also connects the SCC to the TSA and enables the grant mechanism chosen in SIMODE.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Reset	0															
R/W	R/W															
Addr	0xAEC															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—								GR1	SC1	R1CS			T1CS		
Reset	0															
R/W	R/W															
Addr	0xAEE															

Figure 20-17. SI Clock Route Register (SICR)

This register is affected by $\overline{\text{HRESET}}$ but is not affected by $\overline{\text{SRESET}}$. Table 20-5 describes the SICR fields.

Table 20-5. SICR Field Descriptions

Bits	Name	Description
0–23	—	Reserved, should be cleared.
24	GR1	Grant support of SCC1. 0 Transmitter does not support the grant mechanism. The grant is always asserted internally. 1 Transmitter supports the grant mechanism as determined by SIMODE[GMa].
25	SC1	SCC1 connection. 0 SCC1 is not connected to the TSA. It is either connected directly to the NMSI pins or is not used. The choice of general-purpose I/O port versus SCC1 functionality is made in the parallel I/O control register; see Chapter 33, “Parallel I/O Ports.” 1 SCC1 is connected to the multiplexed SI. NMSI receive pins can be used for other purposes.
26–28	R1CS	Receive/transmit clock source for SCC1. Ignored when SCC is connected to the TSA (SC1 = 1). 000 BRG1. 001 BRG2. 010 BRG3. 011 BRG4. 100 CLK1 101 CLK2 110 CLK3 111 CLK4
29–31	T1CS	

20.2.4.4 SI Command Register (SICMR)

The SI command register (SICMR) is used to swap the SI RAM routing. SICMR commands are valid only when the SI RAM is partitioned for dynamic changes; that is, when SIGMR[RDM] = 0b01. See Section 20.2.3.3, “SI RAM Dynamic Changes.”

Bit	0	1	2	3	4	5	6	7
Field	CSRRa	CSRTa	—					
Reset	0							
R/W	R/W							
Addr	0xAE7							

Figure 20-18. SI Command Register (SICMR)

This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Table 20-6 describes the SICMR fields.

Table 20-6. SICMR Field Descriptions

Bits	Name	Description
0	CSRRa	Change shadow RAM for TDMa receiver/transmitter. Set by the user; cleared by the SI when the swap completes. 0 The shadow RAM is invalid. The shadow RAM can be written to program a new routing. 1 The shadow RAM is valid. The SI swaps the RAMs, taking the new routing from the shadow RAM.
1	CSRTa	
2–7	—	Reserved, should be cleared.

20.2.4.5 SI Status Register (SISTR)

The SI status register (SISTR) indicates which part of the SI RAM is the current-route RAM. The value of SISTR is valid only when the corresponding SICMR bit is clear.

Bit	0	1	2	3	4	5	6	7
Field	CRORa	CROTa	—					
Reset	0							
R/W	R							
Addr	OxAE6							

Figure 20-19. SI Status Register (SISTR)

This register is affected by $\overline{\text{HRESET}}$ but is not affected by $\overline{\text{SRESET}}$. Table 20-7 describes the SISTR fields.

Table 20-7. SISTR Field Descriptions

Bits	Name	Description
0	CRORa	Address of the current route of TDMa receiver. 0 Address 0–127 when SIGMR[RDM] = 01. 1 Address 128–255 when SIGMR[RDM] = 01.
1	CROTa	Address of the current route of TDMa transmitter. 0 Address 256–383 when SIGMR[RDM] = 01. 1 Address 384–511 when SIGMR[RDM] = 01.
2–7	—	Reserved, should be cleared.

20.2.4.6 SI RAM Pointer Register (SIRP)

The SI RAM pointer (SIRP) register, shown in Figure 20-20, indicates the RAM entry currently being serviced. SIRP gives the real-time status location of the SI inside a TDM frame—useful for debugging and synchronizing system activity with the TDM's activity. However, simply reading the status register SISTR is sufficient for most applications.

The user can determine which RAM entry in the SI RAM is in progress, but cannot determine the status within that entry. For example, if the SIRP indicates an SI RAM entry is active, but the entry is programmed to select four contiguous 8-bit time slots of a TDM, it cannot be determined which of the four time slots is in progress. However, SIRP updates the status as soon as the next SI RAM entry begins processing. The value of SIRP changes on serial clock transitions. Before acting on the information in this register, perform two reads to verify the same value is returned.

One of the four strobes can be connected externally to an interrupt pin to generate an interrupt on a particular SI RAM entry to start or stop TSA execution.

The pointers in SIRP indicate the SI RAM entry word offset that is in progress.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—		VT2	TaPTR2					—		VT1	TaPTR1				
Reset	0															
R/W	R															
Addr	0XAF0															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—		VR2	RaPTR2					—		VR1	RaPTR1				
Reset	0															
R/W	R															
Addr	0xAF2															

Figure 20-20. SI RAM Pointer Register (SIRP)

This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Table 20-8 describes SIRP fields.

Table 20-8. SIRP Field Descriptions

Bits	Name	Description
0–1, 8–9, 16–17, 24–25	—	Reserved, should be cleared.

Table 20-8. SIRP Field Descriptions (continued)

Bits	Name	Description
2, 10, 18, 26	VR*n*, VT*n*	Valid if set. Knowing whether an entry is valid (active) helps when the PTR value is zero. The V bits eliminate having to read both SIRP and SISTR.
3-7, 11–15, 19–23, 27–31	RaPTR*n*, TaPTR*n*	Transmit/receive SI RAM entry pointers. Incremented by one for each entry processed. These 5-bit pointers' values range from 0–31, corresponding to 32 SI RAM entries, although the entire range may not be used. For instance, if SIRAM[LST] is set in the fifth entry, the pointer reflects values 0–4. When the SI processes the fifth, the pointer returns to 0. Pointer values are described in Table 20-9, and are based on SIGMR[RDM].

Table 20-9 describes the pointer values as affected by SIGMR[RDM].

Table 20-9. SIRP Pointer Values

RDM	Configuration
00	RaPTR1/TaPTR1 point to the first 32 entries and RaPTR2/TaPTR2 point to the second 32 entries. RaPTR*n* and TaPTR*n* point to the active Rx and Tx entries, respectively. When the SI services entries 1–32, RaPTR1/TaPTR1 is incremented and RaPTR2/TaPTR2 is continuously cleared. Conversely, when the SI services entries 33–64, RaPTR1/TaPTR1 is continuously cleared and RaPTR2/TaPTR2 is incremented.
01	For the receiver, whether RaPTR1 or RaPTR2 is used depends on which portion of the SI Rx RAM is active (V-bit set). Likewise, whether TaPTR1 or TaPTR2 is used depends on which portion of the Tx RAM is active. • •If VR1 = 1, RaPTR1 points to the active RXa entry. The Rx address block is 0–127; SISTR[CRORa] = 0. • •If VR2 = 1, RaPTR2 points to the active RXa entry. The Rx address block is 128–255; SISTR[CRORa] = 1. • •If VT1 = 1, TaPTR1 points to the active TXa entry. The Tx address block is 256–383; SISTR[CROTa] = 0. • •If VT2 = 1, TaPTR2 points to the active TXa entry. The Tx address block is 384–511; SISTR[CROTa] = 1.
1x	Reserved

20.3 NMSI Configuration

The SI supports a non-multiplexed serial interface (NMSI) mode for the SCC and SMCs. The decision of whether to connect the SCC to the NMSI is made in the SICR; the SMCs are connected through SIMODE. The SCC or SMC can be connected to the NMSI, regardless of the other channels connected to a TDM channel. However, NMSI pins can be multiplexed with other functions at the parallel I/O lines. Therefore, if a combination of TDM and NMSI channels are used, the decision of which SMC to connect and where to connect them should be made by consulting the pinout in ."

The clocks that are provided to the SCC and SMCs are derived from four internal baud rate generators and external CLK pins. There are two main advantages to this bank-of-clocks approach. First, the SCC or SMC is not forced to choose its clock from a predefined pin or baud rate generator. Second, if a group of receivers and transmitters need the same clock rate they can share the same pin, leaving other pins available other functions and minimizing the potential skew between multiple clock sources.

The baud rate generators also make their clocks available to external logic, regardless of whether the BRGs are being used by the SCC or an SMC. The BRGO*n* pins are multiplexed with other functions, so all BRGO*n* pins may not always be available.

The following restrictions apply to the bank-of-clocks mapping:

- Only eight of the twelve clock sources can be connected to the SCC receiver or transmitter.
- The SMC transmitter must have the same clock source as the receiver when connected to the NMSI.

Once the clock source is selected, the clock is given an internal name. For the SCC, the name is RCLK1 and TCLK1 and for the SMCs, the name is simply SMCLK*x*. These internal names are used only in NMSI mode to specify the clock that is sent to the SCC or SMC. These names do not correspond to physical pins on the MPC855T. Note the internal RCLK1 and TCLK1 can be used as inputs to the DPLL unit, which is inside the SCC1; thus, RCLK1 and TCLK1 are not always required to reflect the actual bit rate on the line.

The clock signals available to each SCC and SMC in NMSI mode are shown in Figure 20-21.

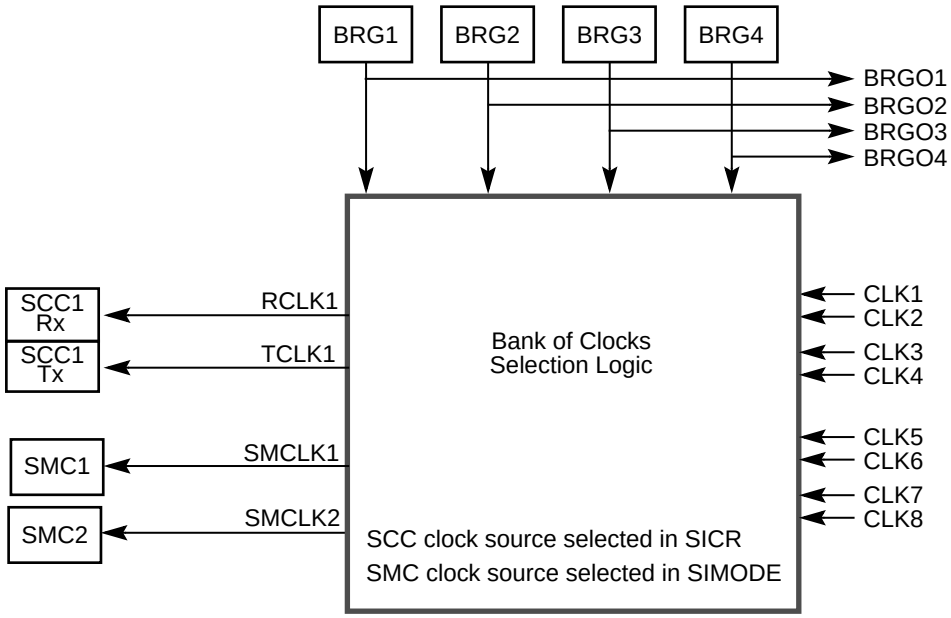


Figure 20-21. Bank-of-Clocks Selection Logic for NMSI

The SCC1 in NMSI mode has its own set of modem control signals:

- TXD1
- RXD1
- TCLK1 ← BRG1–BRG4, CLK1–CLK4
- RCLK1 ← BRG1–BRG4, CLK1–CLK4
- $\overline{\text{RTS1}}$
- $\overline{\text{CTS1}}$

- $\overline{CD1}$

The SMC1 in NMSI mode has its own set of modem control signals:

- SMTXD1
- SMRXD1
- SMCLK1 ← BRG1–BRG4, CLK1–CLK4
- $\overline{SMSYN1}$ (used only in the totally transparent protocol)

The SMC2 in NMSI mode has its own set of modem control signals:

- SMTXD2
- SMRXD2
- SMCLK2 ← BRG1–BRG4, CLK5–CLK8
- $\overline{SMSYN2}$ (used only in the totally transparent protocol)

Unused SCC or SMC signals can be used for other functions or configured for parallel I/O.

20.4 Baud Rate Generators (BRGs)

The CPM contains four independent, identical baud rate generators (BRG) that can be used with the SCC and SMCs. The clocks produced by the BRGs are sent to the bank-of-clocks selection logic, where they can be routed to the SCC and/or SMCs. In addition, the output of a BRG 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
- On-the-fly changes allowed
- Each baud rate generator can be routed to one or more SCC or SMCs
- A 16x divider option allows slow baud rates at high system frequencies
- Each BRG contains an autobaud support option
- Each BRG output can be routed to a pin (BRGO*n*)

Figure 20-22 shows a baud rate generator.

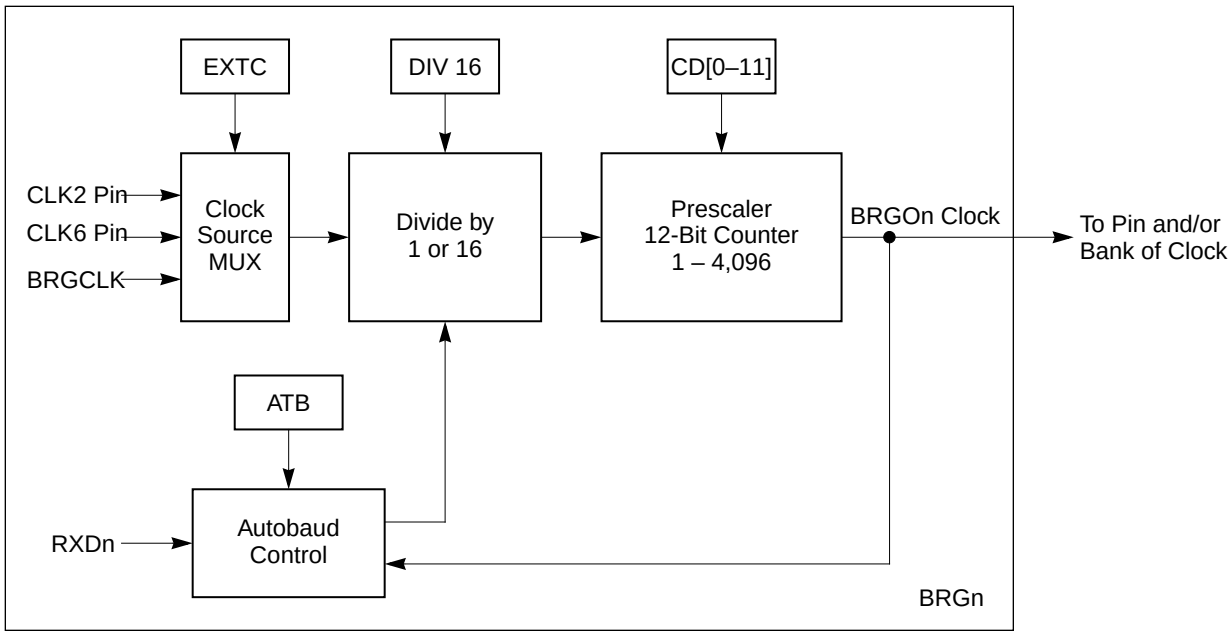


Figure 20-22. Baud Rate Generator (BRG) Block Diagram

The BRG clock source can be BRGCLK, CLK2, or CLK6 (selected in BRGC*n*[EXTC]). The BRGCLK is generated in the MPC855T clock synthesizer specifically for the BRGs, the SPI, and the I^2C internal baud rate generator. Alternatively, the CLK2 and CLK6 pins can be configured as clock sources. These external source options allow flexible baud rate frequency generation, independent of the system frequency. Additionally, CLK2 and CLK6 allow a single external frequency to be the source for multiple BRGs. Note that the CLK2 and CLK6 signals are not synchronized internally before being used by the BRG.

The BRG provides a divide-by-16 option (BRGC*n*[DIV16]) and a 12-bit prescaler (BRGC*n*[CD]) to divide the source clock frequency. The combined source-clock divide factor can be changed on-the-fly; however, two changes should not occur within two source clock periods.

The prescaler output is sent internally to the bank of clocks and can also be output externally on BRGO*n* through either the port A or port B parallel I/O. If the BRG divides the clock by an even value, the transitions of BRGO*n* always occur on the falling edge of the source clock. If the divide factor is odd, the transitions alternate between the falling and rising edges of the source clock. Additionally, the output of the BRG can be sent to the autobaud control block.

20.4.1 Baud Rate Generator Configuration Registers (BRGC*n*)

Each baud rate generator configuration register (BRGC), shown in Figure 20-23, is cleared at reset. A reset disables the BRG and drives the BRGO output clock high. The BRGC can

be written at any time with no need to disable the SCC or external devices that are connected to BRGO. Configuration changes occur at the end of the next BRG clock cycle (no spikes occur on the BRGO output clock). BRGC can be changed on-the-fly; however, two changes should not occur within a time equal to two source clock periods.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—														RST	EN
Reset	0															
R/W	R/W															
Addr	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															
R/W	R/W															
Addr	0x9F2 (BRGC1), 0x9F6 (BRGC2), 0x9FA (BRGC3), 0x9FE (BRGC4)															

Figure 20-23. Baud Rate Generator Configuration Registers (BRGCn)

These registers are affected by $\overline{\text{HRESET}}$ but are not affected by $\overline{\text{SRESET}}$. Table 20-10 describes the BRGC*n* fields.

Table 20-10. BRGC*n* Field Descriptions

Bits	Name	Description
0–13	—	Reserved, should be cleared.
14	RST	Reset BRG. Performs a software reset of the BRG identical to that of an external reset. A reset disables the BRG and drives BRGO high. This is externally visible only if BRGO is connected to the corresponding parallel I/O pin. 0 Enable the BRG. 1 Reset the BRG (software reset).
15	EN	Enable BRG count. Used to dynamically stop the BRG from counting—useful for low-power modes. 0 Stop all clocks to the BRG. 1 Enable clocks to the BRG.
16–17	EXTC	External clock source. Selects the BRG input clock. 00 BRGCLK (internal clock generated by the clock synthesizer in the SIU). 01 CLK2 10 CLK6 11 Reserved.
18	ATB	Autobaud. Selects autobaud operation for BRG1 on the corresponding RXD1. ATB must remain zero until the SCC receives the three Rx clocks. Then the user must set ATB to obtain the correct baud rate. After the baud rate is obtained and locked, it is indicated by setting AB in the UART event register. See Section 20.4.2, “Autobaud Operation on the SCC UART.” 0 Normal operation of the BRG. 1 When RXD goes low, the BRG determines the length of the start bit and synchronizes the BRG to the actual baud rate.

Table 20-10. BRGC*n* Field Descriptions (continued)

Bits	Name	Description
19–30	CD	Clock divider. CD presets an internal 12-bit counter that is decremented at the DIV16 output rate. When the counter reaches zero, it is reloaded with CD. CD = 0xFFF produces the minimum clock rate for BGRO (divide by 4,096); CD = 0x000 produces the maximum rate (divide by 1). 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. See Section 20.4.3, “UART Baud Rate Examples.”
31	DIV16	Divide-by-16. Selects a divide-by-1 or divide-by-16 prescaler before reaching the clock divider. See Section 20.4.3, “UART Baud Rate Examples.” 0 Divide by 1. 1 Divide by 16.

20.4.2 Autobaud Operation on the SCC UART

During the autobaud process, the SCC UART deduces the baud rate of its received character stream by examining the received pattern and its timing. A built-in autobaud control function automatically measures the length of a start bit and modifies the baud rate accordingly.

If the autobaud bit BRGC1[ATB] is set, the autobaud control function starts searching for a low level on the corresponding RXD1 input, which it assumes marks 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 source clock rate and then stops with BRGO1 in the low state.

When RXD1 goes high again, the autobaud control block rewrites BRGC1[CD, DIV16] to the divide ratio found, which at high baud rates may not be exactly the final rate desired (for example, 56,600 may result rather than 57,600). An interrupt can be enabled in the UART SCC event register to report that the autobaud controller rewrote BRGC1. The interrupt handler can then adjust BRGC1[CD, DIV16] (see Table 20-11) for accuracy before the first character is fully received, ensuring that the UART recognizes all characters.

After a full character is received, the software can verify that the character matches a predefined value (such as ‘a’ or ‘A’). Software should then check for other characters (such as ‘t’ or ‘T’) and program the preferred parity mode in the UART’s protocol-specific mode register (PSMR).

Note that SCC1 must be programmed to UART mode and select the 16× option for TDCR and RDCR in the general SCC mode register (GSMR_L). Input frequencies such as 1.8432, 3.68, 7.36, and 14.72 MHz should be used.

Also, to detect an autobaud lock and generate an interrupt, the SCC must receive three full Rx clocks from the BRG before the autobaud process begins. To do this, first clear BRGC1[ATB] and enable the BRG receive clock to the highest frequency. Then,

immediately before the autobaud process starts (after device initialization), set BRGC1[ATB].

20.4.3 UART Baud Rate Examples

For synchronous communication using the internal BRG, the BRGO output clock must not exceed the system frequency divided by 2. So, with a 25-MHz system frequency, the maximum BRGO rate is 12.5 MHz. Program the UART to 16× oversampling when using the SCC as a UART. Rates of 8× and 32× are also available. Assuming 16× oversampling is chosen in the UART, the maximum data rate is 25 MHz ÷ 16 = 1.5625 Mbps. Keeping the above in mind, use the following formula to calculate the bit rate based on a particular BRG configuration for a UART:

```
async baud rate = (BRGCLK or CLK2 or CLK6) ÷ (1 or 16 according to BRGCx[DIV16])
÷ (clock divider + 1) ÷ (8, 16, or 32 according to GSMR_L[TDCR, RDCR] in the
general SCC mode register low)
```

Table 20-11 lists typical bit rates of asynchronous communication. Notice that here the internal clock rate is assumed to be 16× the baud rate; that is, GSMR_L[TDCR] = GSMR_L[RDCR] = 0b10.

Table 20-11. Typical Baud Rates for Asynchronous Communication

Baud Rate	System Frequency								
	20 MHz			25 MHz			24.5760 MHz		
	Div16	CD	Actual Frequency	Div16	CD	Actual Frequency	Div16	CD	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

For synchronous communication, the internal clock is identical to the baud rate output. To get the preferred rate, select the system clock according to the following:

```
sync baud rate = (BRGCLK or CLK2 or CLK6) ÷ (1 or 16 according to BRGCx[DIV16])
÷ (clock divider + 1)
```

For example, to get a rate of 64 kbps, the system clock can be 24.96 MHz, DIV16 = 0, and the clock divider = 389.

Chapter 21
Serial Communications Controller

The MPC855T has one serial communications controller (SCC1), which can be configured independently to implement different protocols for bridging functions, routers, and gateways, and to interface with a wide variety of standard WANs, LANs, and proprietary networks. The SCC has many physical interface options such as interfacing to a TDM bus, an ISDN bus, or a standard modem interface.

The SCC is independent from the physical interface, but SCC logic formats and manipulates data from the physical interface. Furthermore, the choice of protocol is independent from the choice of interface. The SCC is described in terms of the protocol it runs. When the SCC is programmed to a certain protocol or mode, it implements functionality that corresponds to parts of the protocol's link layer (layer 2 of the OSI reference model). Many SCC functions are common to protocols of the following controllers:

- UART, described in Chapter 22, "SCC UART Mode."
- HDLC and HDLC bus, described in Chapter 23, "SCC HDLC Mode."
- IrDA or asynchronous HDLC, described in Chapter 25, "SCC Asynchronous HDLC Mode and IrDA."
- AppleTalk/LocalTalk, described in Chapter 24, "SCC AppleTalk Mode."
- BISYNC, described in Chapter 26, "SCC BISYNC Mode."
- Transparent, described in Chapter 28, "SCC Transparent Mode."
- Ethernet, described in Chapter 27, "SCC Ethernet Mode."

Although the selected protocol usually applies to both the SCC transmitter and receiver, one half of the SCC can run transparent operations while the other half runs a standard protocol (except Ethernet and serial ATM as applicable).

Each Rx and Tx internal clock can be programmed with either an external or internal source. Internal clocks originate from one of four baud rate generators (BRGs) or one of eight external clock pins; see Section 20.2.4.3, "SI Clock Route Register (SICR)," for the SCC's available clock sources. These clocks can be as fast as a 1:2 ratio of the system clock. (For example, the SCC internal clock can run at 12.5 MHz in a 25-MHz system.) However, the SCC's ability to support a sustained bit stream depends on the protocol as well as other factors. See Appendix B, "Serial Communications Performance."

Associated with the SCC is a digital phase-locked loop (DPLL) for external clock recovery, which supports NRZ, NRZI, FM0, FM1, Manchester, and Differential Manchester. If the clock recovery function is not required (that is, synchronous communication), then the DPLL can be disabled, in which case only NRZ and NRZI are supported.

The SCC can be connected to its own set of pins on the MPC855T. This configuration is called the non-multiplexed serial interface (NMSI) and is described in Chapter 20, “Serial Interface.” Using NMSI, the SCC can support standard modem interface signals, $\overline{\text{RTS}}$, $\overline{\text{CTS}}$, and $\overline{\text{CD}}$, through the port C pins and the CPM interrupt controller (CPIC). If required, software and additional parallel I/O lines can be used to support additional handshake signals. Figure 21-1 shows the SCC block diagram.

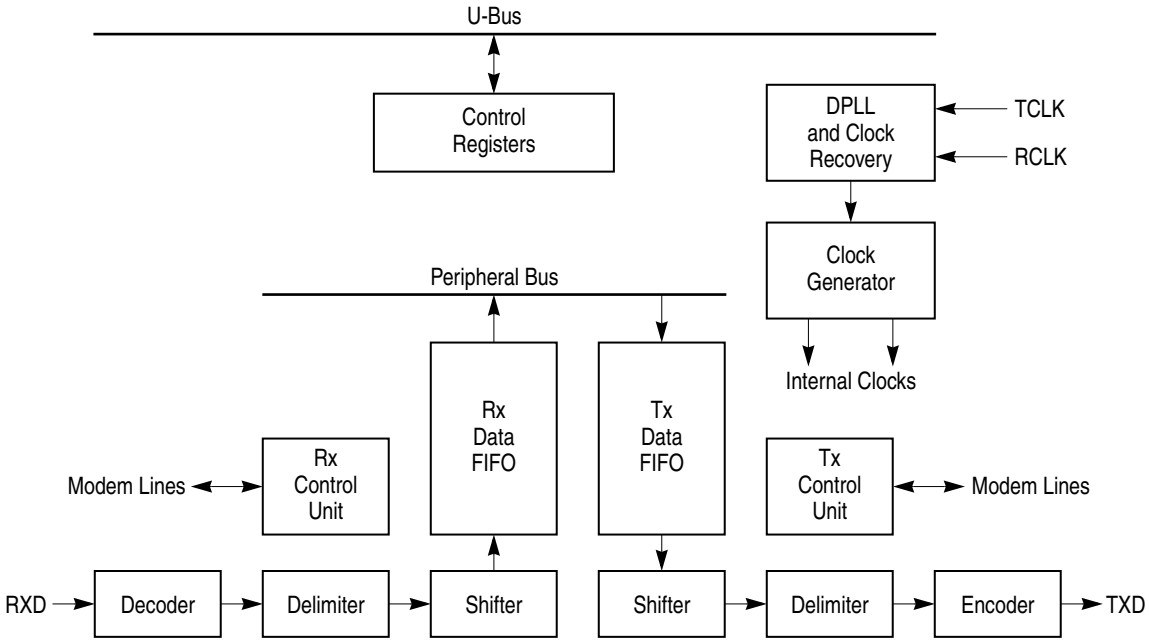


Figure 21-1. SCC Block Diagram

21.1 Features

The following is a list of the main SCC features. (Performance figures assume a 25-MHz system clock.)

- Implements HDLC/SDLC, HDLC bus, asynchronous HDLC, BISYNC, synchronous start/stop, asynchronous start/stop (UART), AppleTalk/LocalTalk, and totally transparent protocols
- Supports 10-Mbps Ethernet/IEEE 802.3 (half- or full-duplex)
- Additional protocols can be added in the future through the use of RAM microcodes. Maximum serial clocking rates of 12.5 MHz on a 25-MHz system

- 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 rate for asynchronous communication can be as high as 3.125 Mbps at 25 MHz
- Supports automatic control of the $\overline{\text{RTS}}$, $\overline{\text{CTS}}$, and $\overline{\text{CD}}$ modem signals
- Multi-buffer data structure for receive and send (the number of buffer descriptors (BDs) is limited only by the size of the internal dual-port RAM—8 bytes per BD)
- Deep FIFOs (The SCC1 transmit and receive FIFOs are 32 bytes each.)
- Transmit-on-demand feature decreases time to frame transmission (transmit latency)
- Low FIFO latency option for send and receive in character-oriented and totally transparent protocols
- Frame preamble options
- Full-duplex operation
- Fully transparent option for one half of the SCC (Rx/Tx) while another protocol executes on the other half (Tx/Rx)—except for Ethernet and serial ATM operation
- Echo and local loopback modes for testing

21.2 SCC Registers

The SCC has a general SCC mode register (GSMR), a protocol-specific mode register (PSMR), a data synchronization register (DSR), and a transmit-on-demand register (TODR). The SCC supporting registers are described in the following sections.

21.2.1 General SCC Mode Register (GSMR)

The SCC contains a general SCC mode register (GSMR) that defines options common to the SCC regardless of the protocol. GSMR_L contains the low-order 32 bits; GSMR_H, shown in Figure 21-2, contains the high-order 32 bits. Some GSMR operations are described in later sections. These registers are affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															GDE
Reset	0															
R/W	R/W															
Addr	0xA04 (GSMR_H1)															
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															
R/W	R/W															
Addr	0xA06 (GSMR_H1)															

Figure 21-2. GSMR_H—General SCC Mode Register (High Order)

Table 21-1 describes GSMR_H fields.

Table 21-1. GSMR_H Field Descriptions

Bit	Name	Description
0–14	—	Reserved, should be cleared.
15	GDE	Glitch detect enable. Determines whether the SCC searches for glitches on the external Rx and Tx serial clock lines. Regardless of the GDE setting, a Schmitt trigger on the input lines is used to reduce signal noise. 0 No glitch detection. Clear GDE if the external serial clock exceeds the limits of glitch detection logic (6.25 MHz assuming a 25-MHz system clock), if an internal BRG supplies the SCC clock, or if external clocks are used and glitch detection matters less than power consumption. 1 Glitches can be detected and reported as maskable interrupts in the SCC event register (SCCE).
16–17	TCRC	Transparent CRC (valid for totally transparent channel only). Selects the frame checking provided on transparent channels of the SCC (either the receiver, transmitter, or both, as defined by TTX and TRX). Although this configuration selects a frame check type, the decision to send the frame check is made in the TxBD. Thus, frame checks are not needed in transparent mode and frame check errors generated on the receiver can be ignored. 00 16-bit CCITT CRC (HDLC). (X16 + X12 + X5 + 1). 01 CRC16 (BISYNC). (X16 + X15 + X2 + 1). 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.
18	REVD	Reverse data (valid for a totally transparent channel only) 0 Normal operation. 1 Reverses the bit order for totally transparent channels on the SCC (either the receiver, transmitter, or both, as defined by TTX and TRX) and sends the msb of each byte first. Section 26.11, “BISYNC Mode Register (PSMR),” describes reversing bit order in a BISYNC protocol.

Table 21-1. GSMR_H Field Descriptions (continued)

Bit	Name	Description
19, 20	TRX, TTX	Transparent receiver/transmitter. The receiver, transmitter, or both can use totally transparent operation, regardless of GSMR_L[MODE]. For example, to configure the transmitter as a UART and the receiver for totally transparent operation, set MODE = 0b0100 (UART), TTX = 0, and TRX = 1. 0 Normal operation. 1 The receiver/transmitter uses totally transparent mode, regardless of the protocol chosen in GSMR_L[MODE]. For full-duplex totally transparent operation, set both TTX and TRX. Note that the SCC cannot operate half in Ethernet or serial ATM mode and half in transparent mode. For example, if MODE = 0b1100 (Ethernet), erratic operation occurs unless TTX = TRX.
21, 22	CDP, CTSP	$\overline{CD}$/$\overline{CTS}$ pulse. If the SCC is used in the TSA and is programmed in transparent mode, set CTSP and refer to Section 28.4.2, “Synchronization and the TSA,” for options on programming CDP. 0 Normal operation (envelope mode). $\overline{CD}$/$\overline{CTS}$ should envelope the frame. Negating $\overline{CD}$/$\overline{CTS}$ during reception causes a $\overline{CD}$/$\overline{CTS}$ lost error. 1 Pulse mode. Synchronization occurs when $\overline{CD}$/$\overline{CTS}$ is asserted; further $\overline{CD}$/$\overline{CTS}$ transitions do not affect reception.
23, 24	CDS, CTSS	$\overline{CD}$/$\overline{CTS}$ sampling. Determine synchronization characteristics of $\overline{CD}$ and $\overline{CTS}$. If the SCC is in transparent mode and is used in the TSA, CDS and CTSS must be set. Also, CDS and CTSS must be set for loopback testing in transparent mode. 0 $\overline{CD}$/$\overline{CTS}$ is assumed to be asynchronous with data. It is internally synchronized by the SCC, then data is received ($\overline{CD}$) or sent ($\overline{CTS}$) after several clock delays. 1 $\overline{CD}$/$\overline{CTS}$ is assumed to be synchronous with data, which speeds up operation. $\overline{CD}$ or $\overline{CTS}$ must transition while the Rx/Tx clock is low, at which time, the transfer begins. Useful for connecting MPC855T in transparent mode since the $\overline{RTS}$ of one MPC855T can connect directly to the $\overline{CD}$/$\overline{CTS}$ of another.
25	TFL	Transmit FIFO length. 0 Normal operation. The SCC transmit FIFO is 32 bytes. 1 The Tx FIFO is 1 byte. This option is used with character-oriented protocols, such as UART, to ensure a minimum FIFO latency at the expense of performance.
26	RFW	Rx FIFO width. 0 Receive FIFO is 32 bits wide for maximum performance; the Rx FIFO is 32 bytes for the SCC. 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 and is recommended for high-performance transparent protocols. 1 Low-latency operation. The receive FIFO is 8 bits wide, reducing the Rx FIFO to a quarter its normal size. This allows data to be written to the buffer as soon as 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. However, it must not be used with HDLC, HDLC Bus, AppleTalk, or Ethernet because it causes erratic behavior.
27	TXSY	Transmitter synchronized to the receiver. 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 bit stream is synchronized to the receiver. Additionally, if RSYN = 1, transmission in totally transparent mode does not occur until the receiver synchronizes with the bit stream and $\overline{CTS}$ is asserted to the SCC. Assuming $\overline{CTS}$ is asserted, transmission begins 8 clocks after the receiver starts receiving data.

Table 21-1. GSMR_H Field Descriptions (continued)

Bit	Name	Description
28–29	SYNL	Sync length (BISYNC and transparent mode only). See the data synchronization register (DSR) definition in the BISYNC (Section 26.9, “Sending and Receiving the Synchronization Sequence”) and totally transparent (Section 28.4.2.1, “In-line Synchronization Pattern”) chapters. 00 An external sync ($\overline{CD}$) is used instead of the sync pattern in the DSR. 01 4-bit sync. The receiver synchronizes on a 4-bit sync pattern stored in the DSR. This sync and additional syncs can be stripped by programming the SCC’s parameter RAM for character recognition. 10 8-bit sync. Should be chosen along with the BISYNC protocol to implement mono-sync. The receiver synchronizes on an 8-bit sync pattern in the DSR. 11 16-bit sync. Also called BISYNC. The receiver synchronizes on a 16-bit sync pattern stored in the DSR.
30	RTSM	$\overline{RTS}$ mode. Determines whether flags or idles are to be sent. Can be changed on-the-fly. 0 Send idles between frames as defined by the protocol and the TEND bit. $\overline{RTS}$ is negated between frames (default). 1 Send flags/syncs between frames according to the protocol. $\overline{RTS}$ is always asserted whenever the SCC is enabled.
31	RSYN	Receive synchronization timing (totally transparent mode only). 0 Normal operation. 1 If CDS = 1, $\overline{CD}$ should be asserted on the second bit of the Rx frame rather than on the first.

Figure 21-3 shows GSMR_L.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—	EDGE		TCI	TSNC		RINV	TINV	TPL			TPP		TEND	TDCR	
Reset	0															
R/W	R/W															
Addr	0xA00 (GSMR_L1)															
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															
R/W	R/W															
Addr	0xA02 (GSMR_L1)															

Figure 21-3. GSMR_L—General SCC Mode Register (Low Order)

Table 21-2 describes GSMR_L fields.

Table 21-2. GSMR_L Field Descriptions

Bit	Name	Description
0	—	Reserved, should be cleared.
1–2	EDGE	Clock edge. Determines the clock edge the DPLL uses to adjust the receive sample point due to jitter in the received signal. Ignored in UART protocol or if the 1x clock mode is selected in RDCR. 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 to change the sample point. 10 Negative edge. Only the negative edge of the received signal is used to change the sample point. 11 No adjustment is made to the sample point.
3	TCI	Transmit clock invert. 0 Normal operation. 1 Before it is used, the internal Tx clock (TCLK) is inverted by the SCC so it can clock data out one-half clock earlier (on the rising rather than the falling edge). In this case, the SCC offers a minimum and maximum rising clock edge-to-data specification. Data output by the SCC after the rising edge of an external Tx clock can be latched by the external receiver one clock cycle later on the next rising edge of the same Tx clock. Recommended for Ethernet, HDLC, and transparent operation when clock rates exceed 8 MHz to improve data setup time for the external transceiver.
4–5	TSNC	Transmit sense. Determines the amount of time the internal carrier sense signal stays active after the last transition on RXD, indicating that the line is free. For instance, AppleTalk can use TSNC to avoid a spurious CS-changed (SCCE[DCC]) interrupt that would otherwise occur during the frame sync sequence before the opening flags. If RDCR is configured to 1× clock 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 smaller number. 00 Infinite. Carrier sense is always active (default). 01 14- or 6.5-bit times as determined by RDCR. 10 4- or 1.5-bit times as determined by RDCR (normally for AppleTalk). 11 3- or 1-bit times as determined by RDCR.
6	RINV	DPLL Rx input invert data. Must be zero in HDLC bus mode or asynchronous UART mode. 0 Do not invert. 1 Invert data before sending it to the DPLL for reception. Used to produce FM1 from FM0 and NRZI space from NRZI mark or to invert the data stream in regular NRZ mode.
7	TINV	DPLL Tx input invert data. Must be zero in HDLC bus mode. 0 Do not invert. 1 Invert data before sending it to the DPLL for transmission. Used to produce FM1 from FM0 and NRZI space from NRZI mark and to invert the data stream in regular NRZ mode. In T1 applications, setting TINV and TEND creates a continuously inverted HDLC data stream.
8–10	TPL	Tx preamble length. Determines the length of the preamble configured by the TPP bits. 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.

Table 21-2. GSMR_L Field Descriptions (continued)

Bit	Name	Description
11–12	TPP	Tx preamble pattern. Determines what, if any, bit pattern should precede each Tx frame. The preamble pattern is sent before the first flag/sync of the frame. TPP is ignored in UART mode. The preamble length is programmed in TPL; the preamble pattern is typically sent to a receiving station that uses a DPLL for clock recovery. The receiving DPLL uses the regular preamble pattern 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.
13	TEND	Transmitter frame ending. Intended for NRZI transmitter encoding of the DPLL. TEND determines whether TXD should idle in a high state or in an encoded ones state (high or low). It can, however, be used with other encodings besides NRZI. 0 Default operation. TXD is encoded only when data is sent, including the preamble and opening and closing flags/syncs. When no data is available to send, the signal is driven high. 1 TXD is always encoded, even when idles are sent.
14–15	TDCR	Transmitter/receiver DPLL clock rate. If the DPLL is not used, choose 1× mode except in asynchronous UART mode where 8×, 16×, or 32× must be chosen. TDCR should match RDCR in most applications to allow the transmitter and receiver to use the same clock source. If an application uses the DPLL, the selection of TDCR/RDCR depends on the encoding/decoding. If communication is synchronous, select 1×. FM0/FM1, Manchester, and Differential Manchester require 8×, 16×, or 32×. If NRZ- or NRZI-encoded communication is asynchronous (that is, clock recovery required), select 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 encodings/decodings are allowed. 01 8× clock mode. 10 16× clock mode. Normally chosen for UART and AppleTalk. 11 32× clock mode.
16–17	RDCR	
18–20	RENC	Receiver decoding/transmitter encoding method. Select NRZ if DPLL is not used. RENC should equal TENC in most applications. However, do not use this internal DPLL for Ethernet. 000 NRZ (default setting if DPLL is not used). Required for UART (synchronous or asynchronous). 001 NRZI Mark (set RINV/TINV also for NRZI space). 010 FM0 (set RINV/TINV also for FM1). 011 Reserved. 100 Manchester. 101 Reserved. 110 Differential Manchester (Differential Bi-phase-L). 111 Reserved.
21–23	TENC	

Table 21-2. GSMR_L Field Descriptions (continued)

Bit	Name	Description
24–25	DIAG	Diagnostic mode. 00 Normal operation, $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ are under automatic control. Data is received through RXD and transmitted through TXD. The SCC uses modem signals to enable or disable transmission and reception. These timings are shown in Section 21.4.4, “Controlling SCC Timing with RTS, CTS, and CD.” 01 Local loopback mode. Transmitter output is connected internally to the receiver input, while the receiver and the transmitter operate normally. The value on RXD is ignored. If enabled, data appears on TXD, or the parallel I/O registers can be programmed to make TXD high. $\overline{\text{RTS}}$ can also be programmed to be disabled in the appropriate parallel I/O register. The transmitter and receiver must share the same clock source, but separate CLK*x* pins can be used if connected to the same external clock source. If external loopback is preferred, program DIAG for normal operation and externally connect TXD and RXD. Then, physically connect the control signals ($\overline{\text{RTS}}$ connected to $\overline{\text{CD}}$, and $\overline{\text{CTS}}$ grounded) or set the parallel I/O registers so $\overline{\text{CD}}$ and $\overline{\text{CTS}}$ are permanently asserted to the SCC by configuring the associated $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ pins as general-purpose I/O. 10 Automatic echo mode. The transmitter automatically resends received data bit-by-bit using the Rx clock provided. The receiver operates normally and receives data if $\overline{\text{CD}}$ is asserted. $\overline{\text{CTS}}$ is ignored. 11 Loopback and echo mode. Loopback and echo operation occur simultaneously. $\overline{\text{CD}}$ and $\overline{\text{CTS}}$ are ignored. See the loopback bit description above for clocking requirements. For TDM operation, the diagnostic mode is selected by SIMODE[SDMa]; see Section 20.2.4.2, “SI Mode Register (SIMODE).”
26	ENR	Enable receive. Enables the receiver hardware state machine for the SCC. 0 The receiver is disabled and data in the Rx FIFO is lost. If ENR is cleared during reception, the receiver aborts the current character. 1 The receiver is enabled. ENR can be set or cleared, regardless of whether serial clocks are present. Section 21.4.7, “Reconfiguring the SCC,” describes how to disable/enable the SCC. Note also these other tools provided for controlling SCC reception: the ENTER HUNT MODE and CLOSE RXBD commands, and RxBD[E].
27	ENT	Enable transmit. Enables the transmitter hardware state machine for the SCC. 0 The transmitter is disabled. If ENT is cleared during transmission, the current character is aborted and TXD returns to the idle state. Data already in the Tx shift register is not sent. 1 The transmitter is enabled. ENT can be set or cleared, regardless of whether serial clocks are present. Section 21.4.7, “Reconfiguring the SCC,” describes how to disable/enable the SCC. Note also these other tools provided for controlling SCC transmission: the STOP TRANSMIT, GRACEFUL STOP TRANSMIT, and RESTART TRANSMIT commands, the freeze option and $\overline{\text{CTS}}$ flow control option in UART mode, and TxBD[R].
28–31	MODE	Channel protocol mode. See also GSMR_H[TTX, TRX]. 0000 HDLC 0001 Reserved 0010 AppleTalk/LocalTalk 0011 SS7—reserved for RAM microcode 0100 UART 0101 0110 0111 V.14—reserved for RAM microcode 1000 BISYNC 1001 DDCMP—reserved for RAM microcode 101x Reserved 1100 Ethernet All others reserved.

21.2.2 Protocol-Specific Mode Register (PSMR)

The protocol implemented by the SCC is selected by its GSMR_L[MODE]. The SCC has an additional protocol-specific mode register (PSMR) for configurations specific to the chosen protocol. The PSMR fields are described in the specific chapters that describe each protocol. These registers are affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

21.2.3 Data Synchronization Register (DSR)

The SCC has a data synchronization register (DSR) that specifies the pattern used for frame synchronization. The programmed value for DSR depends on the protocol:

- UART—DSR is used to configure fractional stop bit transmission.
- BISYNC and transparent—DSR should be programmed with the sync pattern.
- Ethernet—DSR should be programmed with 0xD555.
- HDLC—At reset, DSR defaults to 0x7E7E (two HDLC flags), so it does not need to be written.

This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Figure 21-4 shows the sync fields.

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															
Addr	0xA0E (DSR1)															

Figure 21-4. Data Synchronization Register (DSR)

21.2.4 Transmit-on-Demand Register (TODR)

In normal operation, if no frame is being sent by the SCC, the CP periodically polls the R bit of the next TxBD to see if a new frame/buffer is requested. Depending on the SCC configuration, this polling occurs every 8–32 serial Tx clocks. The transmit-on-demand option, selected in the transmit-on-demand register (TODR) shown in Figure 21-5, shortens the latency of the Tx buffer/frame and is useful in LAN-type protocols where maximum interframe gap times are limited by the protocol specification.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	TOD	—														
Reset	0															
R/W	R/W															
Addr	0xA0C (TODR1)															

Figure 21-5. Transmit-on-Demand Register (TODR)

The CP can be configured to begin processing a new frame/buffer without waiting the normal polling time by setting TODR[TOD] after TxBD[R] is set. Because this feature favors the specified TxBD, it may affect servicing of the FIFOs of other CPM controllers. Therefore, transmitting on demand should only be used when a high-priority TxBD has been prepared and enough time has passed since the last SCC transmission. Table 21-3 describes TODR fields. This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

Table 21-3. TODR Field Descriptions

Bits	Name	Description
0	TOD	Transmit on demand. 0 Normal operation. 1 The CP gives high priority to the current TxBD and begins sending the frame without waiting the normal polling time to check TxBD[R]. TOD is cleared automatically after one serial clock, but transmitting on demand continues until an unprepared (R = 0) BD is reached. TOD does not need to be set again if new TxBDs are added to the BD table as long as older TxBDs are still being processed. New TxBDs are processed in order. The first bit of the frame is typically clocked out 5-6 bit times after TOD is set.
1–15	—	Reserved, should be cleared.

21.3 SCC Buffer Descriptors (BDs)

Data associated with the SCC is stored in buffers and each buffer is referenced by a buffer descriptor (BD) that can reside anywhere in dual-port RAM. The total number of 8-byte BDs is limited only by the size of the dual-port RAM (128 BDs/1 Kbyte). These BDs are shared among all serial controllers—SCC, SMCs, SPI, and I^2C. The user defines how the BDs are allocated among the controllers.

Each 64-bit BD has the following structure:

- The half word at offset + 0x0 contains status and control bits that control and report on the data transfer. These bits vary from protocol to protocol. The CP updates the status bits after the buffer is sent or received.
- The half word at offset + 0x2 (data length) holds the number of bytes sent or received.
  - For an RxBD, this is the number of bytes the controller writes into the buffer. The CP writes the length after received data is placed into the associated buffer and the buffer closed. In frame-based protocols (but not including SCC transparent operation), this field contains the total frame length, including CRC bytes. Also, if a received frame's length, including CRC, is an exact multiple of MRBLR, the last BD holds no actual data but does contain the total frame length.
  - For a TxBD, this is the number of bytes the controller should send from its buffer. Normally, this value should be greater than zero. The CP never modifies this field.

- The word at offset + 0x4 (buffer pointer) points to the beginning of the buffer in memory (internal or external).
  - For an RxBD, the value must be even.
  - For a TxBD, this pointer can be even or odd.

Shown in Figure 21-6, the format of Tx and Rx BDs is the same in each SCC mode. Only the status and control bits differ for each protocol.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	Status and Control															
Offset + 2	Data Length															
Offset + 4	High-Order Buffer Pointer															
Offset + 6	Low-Order Buffer Pointer															

Figure 21-6. SCC Buffer Descriptors (BDs)

For frame-oriented protocols, a message can reside in as many buffers as necessary. Each buffer has a maximum length of 65,535 bytes. The CP does not assume that all buffers of a single frame are currently linked to the BD table. The CP does assume, however, that the unlinked buffers are provided by the core in time to be sent or received; otherwise, an error condition is reported—an underrun error when sending and a busy error when receiving. Figure 21-7 shows the SCC BD table and buffer structure.

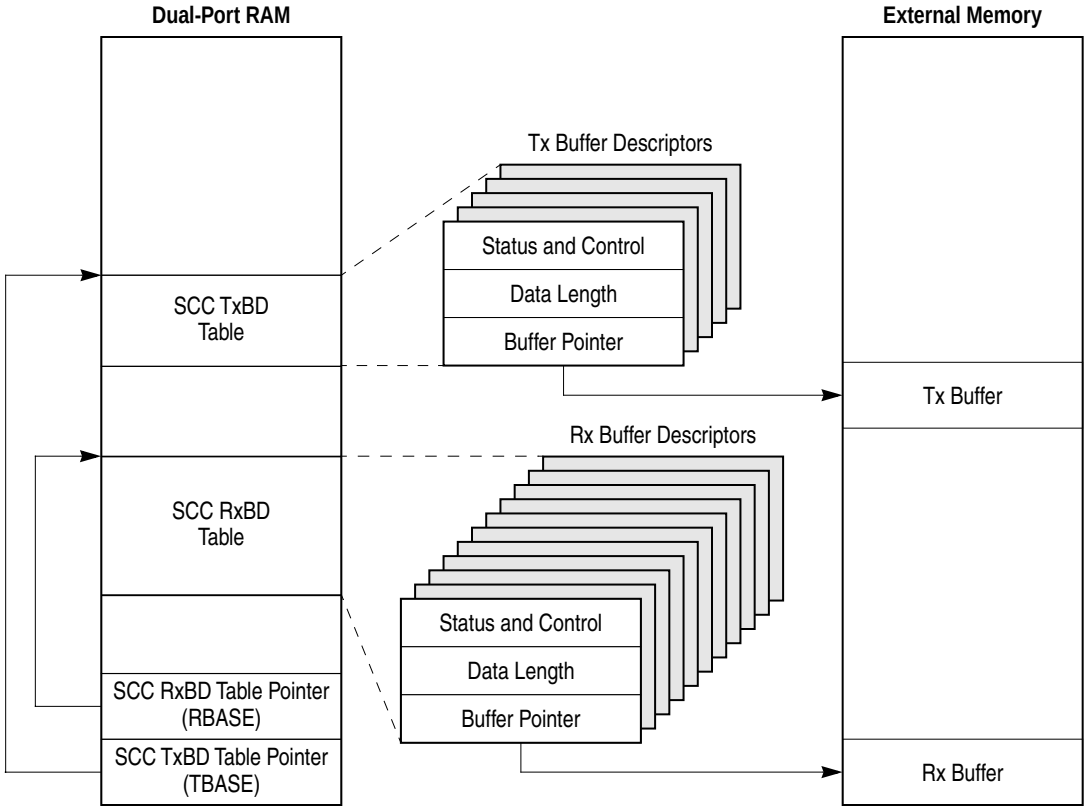


Figure 21-7. SCC Buffer Descriptor and Buffer Structure

In all protocols, BDs can point to buffers in the internal dual-port RAM. However, because internal RAM is used for descriptors, buffers are usually put in external RAM, especially if they are large. Usually, the internal U bus transfers data to the buffer.

The CP processes TxBDs in a straightforward manner. Once the transmit side of the SCC is enabled, it starts with the first BD in that SCC TxBD table. Once the CP detects that the R bit is set in the TxBD, it starts processing the buffer. The CP detects that the BD is ready when it polls the R bit or when the user writes to the TODR. After data from the BD is put in the Tx FIFO, if necessary the CP waits for the next descriptor's R bit to be set before proceeding. Thus, the CP does no look-ahead descriptor processing and does not skip BDs that are not ready. When the CP sees a BD's W bit (wrap) set, it returns to the start of the BD table after this last BD of the table is processed. The CP clears R (not ready) after using a TxBD, which keeps it from being retransmitted before it is confirmed by the core. However, some protocols support a continuous mode (CM), for which R is not cleared (always ready).

The CP uses RxBDs similarly. When data arrives, the CP performs required processing on the data and moves resultant data to the buffer pointed to by the first BD; it continues until the buffer is full or an event, such as an error or end-of-frame detection, occurs. The buffer is then closed; subsequent data uses the next BD. If E = 0, the current buffer is not empty and it reports a busy error. The CP does not move from the current BD until E is set by the

core (the buffer is empty). After using a descriptor, the CP clears E (not empty) and does not reuse a BD until it has been processed by the core. However, in continuous mode (CM), E remains set. When the CP discovers a descriptor's W bit set (indicating it is the last BD in the circular BD table), it returns to the beginning of the table when it is time to move to the next buffer.

21.4 SCC Parameter RAM

The protocol-specific portions of the SCC parameter RAM are discussed in the specific protocol descriptions and the part that is common to all SCC protocols is shown in Table 21-4.

Some parameter RAM values must be initialized before the SCC can be enabled. Other values are initialized or written by the CP. Once initialized, most parameter RAM values do not need to be accessed because most activity centers around the descriptors rather than the parameter RAM. However, if the parameter RAM is accessed, note the following:

- Parameter RAM can be read at any time.
- Tx parameter RAM can be written only when the transmitter is disabled—after a STOP TRANSMIT command and before a RESTART TRANSMIT command or after the buffer/frame finishes transmitting after a GRACEFUL STOP TRANSMIT command and before a RESTART TRANSMIT command.
- Rx parameter RAM can be written only when the receiver is disabled. Note the CLOSE RX BD command does not stop reception, but it does allow the user to extract data from a partially full Rx buffer.
- See Section 21.4.7, "Reconfiguring the SCC."

Table 21-4 shows the parameter RAM map for all SCC protocols. Boldfaced entries must be initialized by the user.

Table 21-4. SCC Parameter RAM Map for All Protocols

Offset [1]	Name	Width	Description
0x00	**RBASE**	Hword	Rx/TxBD table base address—offset from the beginning of dual-port RAM. The BD tables can be placed in any unused portion of the dual-port RAM. Values in RBASE and TBASE should be multiples of eight.
0x02	**TBASE**	Hword	
0x04	**RFCR**	Byte	Rx function code. See Section 21.4.1, "Function Code Registers (RFCR and TFCR)."
0x05	**TFCR**	Byte	Tx function code. See Section 21.4.1, "Function Code Registers (RFCR and TFCR)."

Table 21-4. SCC Parameter RAM Map for All Protocols (continued)

Offset [1]	Name	Width	Description
0x06	**MRBLR**	Hword	Maximum receive buffer length. Defines the maximum number of bytes the CP writes to a receive buffer before it goes to the next buffer. The CP can write fewer bytes than MRBLR if a condition such as an error or end-of-frame occurs. It never writes more bytes than the MRBLR value. Therefore, user-supplied buffers should be no smaller than MRBLR. MRBLR should be greater than zero for all modes. It should be a multiple of 4 for Ethernet and HDLC modes, and in totally transparent mode unless the Rx FIFO is 8-bits wide (GSMR_H[RFW] = 1). Note that although MRBLR is not intended to be changed while the SCC is operating, it can be changed dynamically in a single-cycle, 16-bit move (not two 8-bit cycles). Changing MRBLR has no immediate effect. To guarantee the exact Rx BD on which the change occurs, change MRBLR only while the receiver is disabled. Transmit buffer length is programmed in TxBD[Data Length] and is not affected by MRBLR.
0x08	RSTATE	Word	Rx internal state[3]
0x0C	RIP	Word	Rx internal buffer pointer [2]. The Rx and Tx internal buffer pointers are updated by the SDMA channels to show the next address in the buffer to be accessed.
0x10	RBPTR	Hword	Current RxBD pointer. Points to the current BD being processed or to the next BD the receiver uses when it is idling. After reset or when the end of the BD table is reached, the CP initializes RBPTR to the value in the RBASE. Although most applications do not need to write RBPTR, it can be modified when the receiver is disabled or when no Rx buffer is in use.
0x12	RCOUNT	Hword	Rx internal byte count[2]. The Rx internal byte count is a down-count value initialized with MRBLR and decremented with each byte written by the supporting SDMA channel.
0x14	RTEMP	Word	Rx temp[3]
0x18	TSTATE	Word	Tx internal state[3]
0x1C	TIP	Word	Tx internal buffer pointer[2]. The Rx and Tx internal buffer pointers are updated by the SDMA channels to show the next address in the buffer to be accessed.
0x20	TBPTR	Hword	Current TxBD pointer. Points to the current BD being processed or to the next BD the transmitter uses when it is idling. After reset or when the end of the BD table is reached, the CP initializes TBPTR to the value in the TBASE. Although most applications do not need to write TBPTR, it can be modified when the transmitter is disabled or when no Tx buffer is in use (after a STOP TRANSMIT or GRACEFUL STOP TRANSMIT command is issued and the frame completes its transmission).
0x22	TCOUNT	Hword	Tx internal byte count[2]. A down-count value initialized with TxBD[Data Length] and decremented with each byte read by the supporting SDMA channel.
0x24	TTEMP	Word	Tx temp[3]
0x28	RCRC	Word	Temp receive CRC[2]
0x2C	TCRC	Word	Temp transmit CRC[2]
0x30			Protocol-specific area. (The size of this area depends on the protocol chosen.)

1 From SCC base. SCC base = IMMR + 0x3C00 (SCC1)
2 These parameters need not be accessed for normal operation but may be helpful for debugging.
3 For CP use only

21.4.1 Function Code Registers (RFCR and TFCR)

The SCC has two separate function code registers—one for Rx buffers (RFCR) and one for Tx buffers (TFCR). Function code registers contain the value to appear on AT[1–3] when the associated SDMA channel accesses memory. It also selects the byte-ordering convention. Figure 21-8 shows the register format.

Bit	0	1	2	3	4	5	6	7
Field	—			BO		AT[1–3]		
Reset	0000_0000							
R/W	R/W							
Addr	SCC base + 0x04 (RFCR); SCC base + 0x05 (TFCR)							

Figure 21-8. Function Code Registers (RFCR and TFCR)

Table 21-5 describes RFCR/TFCR fields.

Table 21-5. RFCR /TFCR Field Descriptions

Bits	Name	Description
0–2	—	Reserved, should be cleared.
3–4	**BO**	Byte ordering. Program BO to select the required byte ordering for the buffer. If BO is changed on-the-fly, it takes effect at the beginning of the next frame (Ethernet, HDLC, and transparent) or at the beginning of the next BD. See Appendix A, “Byte Ordering.” 00 Reserved 01 Modified little-endian. 1x Big-endian or true little-endian.
5–7	**AT[1–3]**	Address type. Contains the function code value used during the SDMA channel memory access. Note AT[0] is driven high to identify this SDMA channel access as a DMA type.

21.4.2 Handling SCC Interrupts

SCC interrupts are handled globally by the CPM interrupt controller (CPIC) using the CPM interrupt pending register (CIPR), CPM interrupt mask register (CIMR), and CPM in-service register (CISR), described in Chapter 34, “CPM Interrupt Controller.” Bits in each CPIC register are used to mask, enable, or report individual interrupts in the SCC.

To allow interrupt handling for SCC-specific events, further event, mask, and status registers are provided within the SCC’s internal memory map area; see Table 21-6. Since interrupt events are protocol-dependent, event descriptions are found in the specific protocol chapters.

Table 21-6. SCC*x* Event, Mask, and Status Registers

Register & IMMR Offset	Description
SCCE 0xA10 (SCC1)	SCC event register. This 16-bit register reports events recognized by the SCC. When an event is recognized, the SCC sets its corresponding bit in SCCE, regardless of the corresponding mask bit. When the corresponding event occurs, an interrupt is signaled to the CPIC. Bits are cleared by writing ones (writing zeros has no effect). SCCE is cleared at reset, is affected by HRESET and SRESET, and can be read at any time.
SCCM 0xA14 (SCC1)	SCC mask register. The 16-bit, read/write register allows interrupts to be enabled or disabled using the CPM for specific events in the SCC. An interrupt is generated only if interrupts in the SCC are enabled in the CPIC. If an SCCM bit is zero, the CPM does not proceed with interrupt handling when that event occurs. If an SCCM bit is set, a 1 in the corresponding SCCE bit sets the SCC event bit in CIPR. The SCCM and SCCE bit positions are identical. This register is affected by HRESET and SRESET.
SCCS 0xA17 (SCC1)	SCC status register. This 8-bit, read-only register allows monitoring of the real-time status of RXD. It does not show the real-time status of CTS and CD, which is available in the parallel I/O data registers. Interrupts caused by CTS and CD are described in Section Chapter 33, “Parallel I/O Ports.” This register is affected by HRESET and SRESET.

Follow these steps to handle an SCC interrupt:

1. Once an interrupt occurs, read SCCE to determine the interrupt sources and clear those SCCE bits (in most cases).
2. Process the TxBDs to reuse them if SCCE[TX] or SCCE[TXE] = 1. If the transmit speed is fast or the interrupt delay is long, the SCC may have sent more than one Tx buffer. Thus, it is important to check more than one TxBD during interrupt handling. A common practice is to process all TxBDs in the handler until one is found with its R bit set.
3. Extract data from the RxBD if SCCE[RX], SCCE[RXB], or SCCE[RXF] is set. As with transmit buffers, if the receive speed is fast or the interrupt delay is long, the SCC may have received more than one buffer and the handler should check more than one RxBD. A common practice is to process all RxBDs in the interrupt handler until one is found with its E bit set.
4. Clear CISR[SCC].
5. Execute the **rfi** instruction.

21.4.3 SCC Initialization

SCC initialization requires that a number of registers and parameters be configured after a power-on reset. Regardless of the protocol used, follow these steps:

1. Write the parallel I/O ports to configure and connect the I/O pins to the SCC.
2. Set the SDMA configuration register SDCR[RAID] field to 0b01 (U-bus arbitration priority level 5).
3. Configure the parallel I/O registers to enable RTS, CTS, and CD if these signals are required.

4. If the time-slot assigner (TSA) is used, the serial interface (SI) must be configured. If the SCC is used in NMSI mode, SICR must still be initialized.
5. Write all GSMR bits except ENT or ENR.
6. Write the PSMR.
7. Write the DSR.
8. Initialize the required values for the SCC's parameter RAM.
9. Clear out any current events in SCCE (optional).
10. Write ones to SCCM register to enable interrupts.
11. Clear out any current interrupts in the CIPR (optional).
12. Write the CIMR to enable interrupts to the CPIC.
13. Set GSMR_L[ENT] and GSMR_L[ENR].

Descriptors can have their R or E bits set at any time. Notice that the CPCR does not need to be accessed after a hard reset. The SCC should be disabled and reenabled after any dynamic change to its parallel I/O ports or serial channel physical interface configuration. A full reset can also be implemented using CPCR[RST].

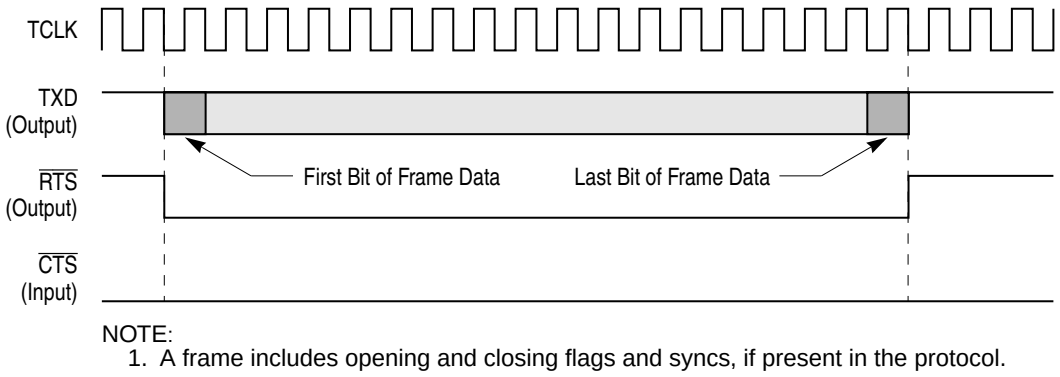
21.4.4 Controlling SCC Timing with $\overline{\text{RTS}}$, $\overline{\text{CTS}}$, and $\overline{\text{CD}}$

When GSMR_L[DIAG] is programmed to normal operation, $\overline{\text{CD}}$ and $\overline{\text{CTS}}$ are controlled by the SCC. In the following subsections, it is assumed that GSMR_L[TCI] is zero, implying normal transmit clock operation.

21.4.4.1 Synchronous Protocols

$\overline{\text{RTS}}$ is asserted when the SCC data is loaded into the Tx FIFO and a falling Tx clock occurs. At this point, the SCC starts sending data once appropriate conditions occur on $\overline{\text{CTS}}$. In all cases, the first data bit is the start of the opening flag, sync pattern, or preamble.

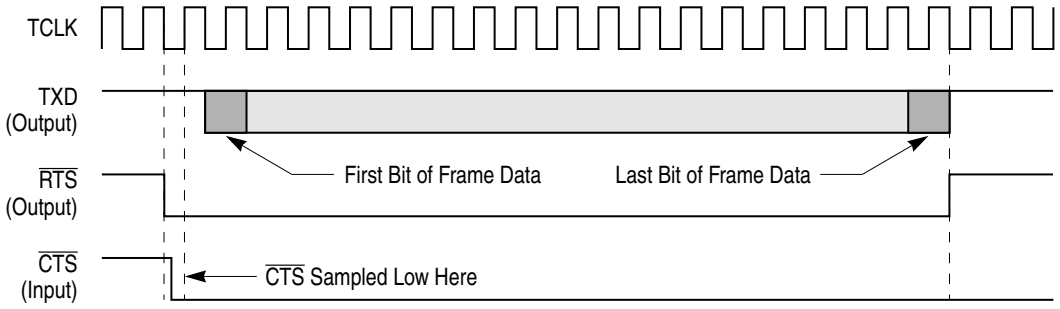
Figure 21-9 shows that the delay between $\overline{\text{RTS}}$ and data is 0 bit times, regardless of GSMR_H[CTSS]. This operation assumes that $\overline{\text{CTS}}$ is already asserted to the SCC or that $\overline{\text{CTS}}$ is reprogrammed to be a parallel I/O line, in which case $\overline{\text{CTS}}$ to the SCC is always asserted. $\overline{\text{RTS}}$ is negated one clock after the last bit in the frame.



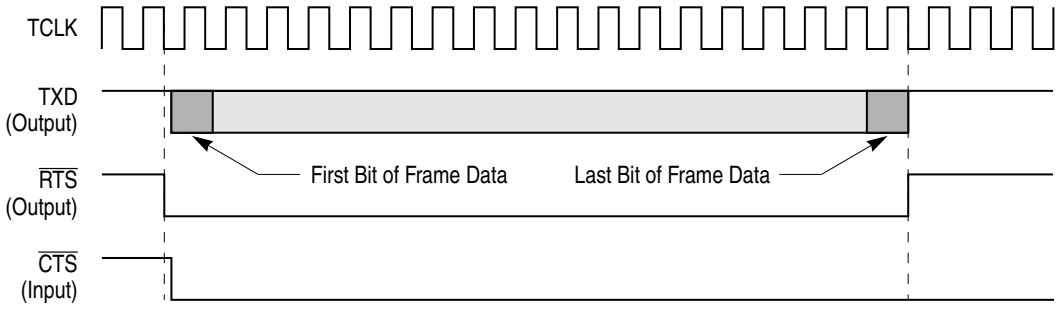
NOTE:
1. A frame includes opening and closing flags and syncs, if present in the protocol.

Figure 21-9. Output Delay from RTS Asserted for Synchronous Protocols

When RTS is asserted, if CTS is not already asserted, delays to the first data bit depend on when CTS is asserted. Figure 21-10 shows that the delay between CTS and the data can be approximately 0.5 to 1 bit times or 0 bit times, depending on GSMR_H[CTSS].



NOTE:
1. GSMR_H[CTSS] = 0. CTSP is a don't care.



NOTE:
1. GSMR_H[CTSS] = 1. CTSP is a don't care.

Figure 21-10. Output Delay from CTS Asserted for Synchronous Protocols

If CTS is programmed to envelope data, negating it during frame transmission causes a CTS lost error. Negating CTS forces RTS high and Tx data to become idle. If GSMR_H[CTSS] is zero, the SCC must sample CTS before a CTS lost is recognized;

otherwise, the negation of $\overline{CTS}$ immediately causes the $\overline{CTS}$ lost condition. See Figure 21-11.

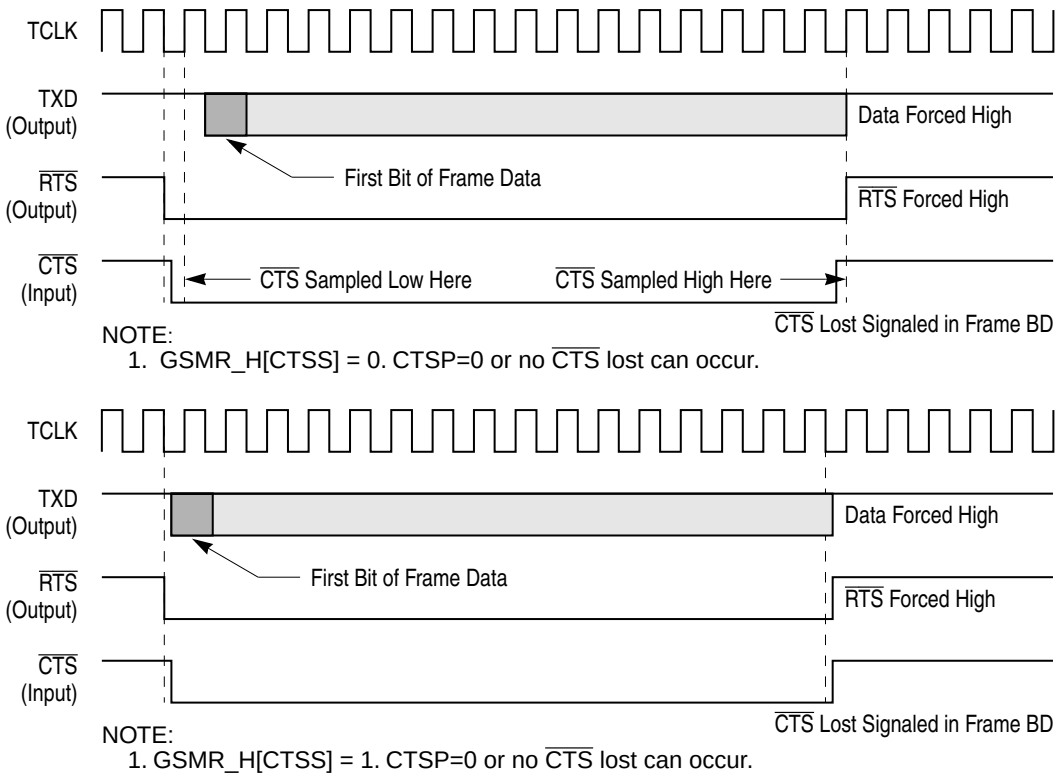


Figure 21-11. $\overline{CTS}$ Lost in Synchronous Protocols

Note that if GSMR_H[CTSS] = 1, $\overline{CTS}$ transitions must occur while the Tx clock is low.

Reception delays are determined by $\overline{CD}$ as shown in Figure 21-12. If GSMR_H[CDS] is zero, $\overline{CD}$ is sampled on the rising Rx clock edge before data is received. If GSMR_H[CDS] is 1, $\overline{CD}$ transitions cause data to be immediately gated into the receiver.

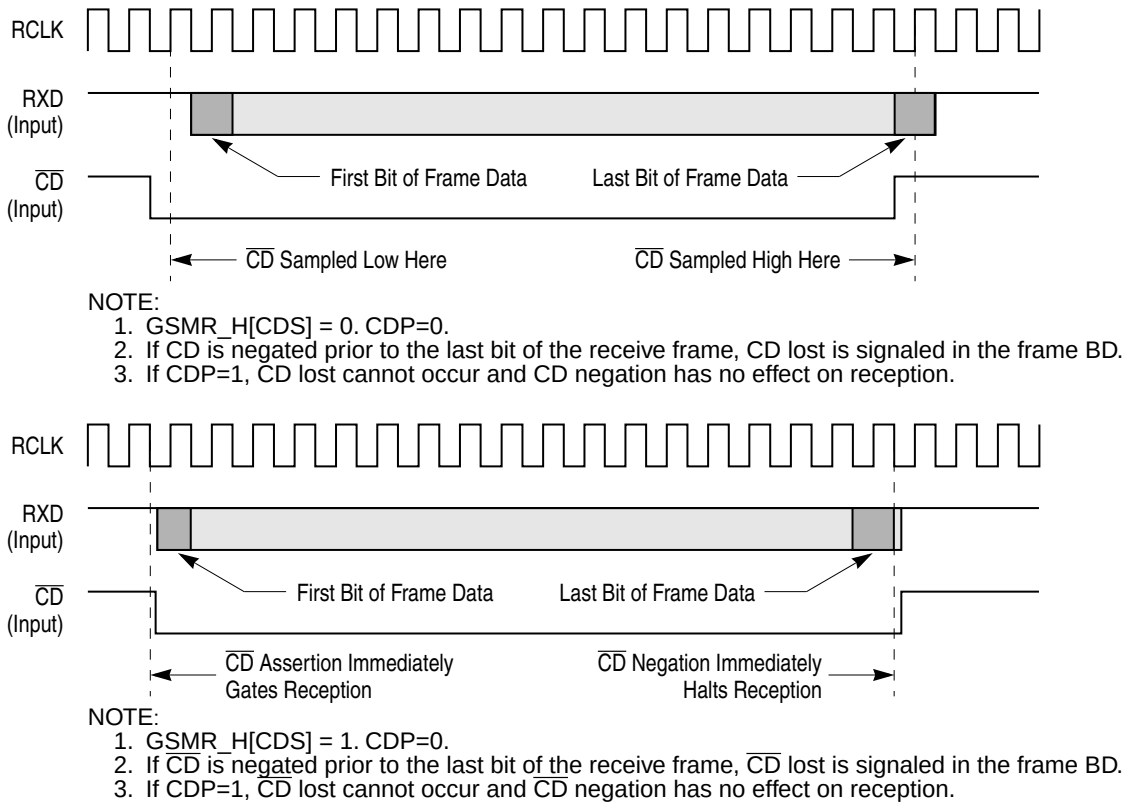


Figure 21-12. Using $\overline{CD}$ to Control Synchronous Protocol Reception

If $\overline{CD}$ is programmed to envelope the data, it must remain asserted during frame transmission or a $\overline{CD}$ lost error occurs. Negation of $\overline{CD}$ terminates reception. If GSMR_H[CDS] is zero, $\overline{CD}$ must be sampled by the SCC before a $\overline{CD}$ lost error is recognized; otherwise, the negation of $\overline{CD}$ immediately causes the $\overline{CD}$ lost condition.

If GSMR_H[CDS] is set, all $\overline{CD}$ transitions must occur while the Rx clock is low.

21.4.4.2 Asynchronous Protocols

In asynchronous protocols, $\overline{RTS}$ is asserted when SCC data is loaded into the Tx FIFO and a falling Tx clock occurs. $\overline{CD}$ and $\overline{CTS}$ can be used to control reception and transmission in the same manner as the synchronous protocols. The first bit sent 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 Chapter 22, “SCC UART Mode.”

- If $\overline{CTS}$ is already asserted when $\overline{RTS}$ is asserted, transmission begins in two additional bit times.
- If $\overline{CTS}$ is not already asserted when $\overline{RTS}$ is asserted and GSMR_H[CTSS] = 0, transmission begins in three additional bit times.
- If $\overline{CTS}$ is not already asserted when $\overline{RTS}$ is asserted and GSMR_H[CTSS] = 1, transmission begins in two additional bit times.

21.4.5 Digital Phase-Locked Loop (DPLL) Operation

The SCC includes a digital phase-locked loop (DPLL) for recovering clock information from a received data stream. For applications that provide a direct clock source to the SCC, the DPLL can be bypassed by selecting 1x mode for GSMR_L[RDCR, TDCR]. If the DPLL is bypassed, only NRZ or NRZI encodings are available. The DPLL must not be used when the SCC is programmed to Ethernet and is optional for other protocols. Figure 21-13 shows the DPLL receiver block; Figure 21-14 shows the transmitter block diagram.

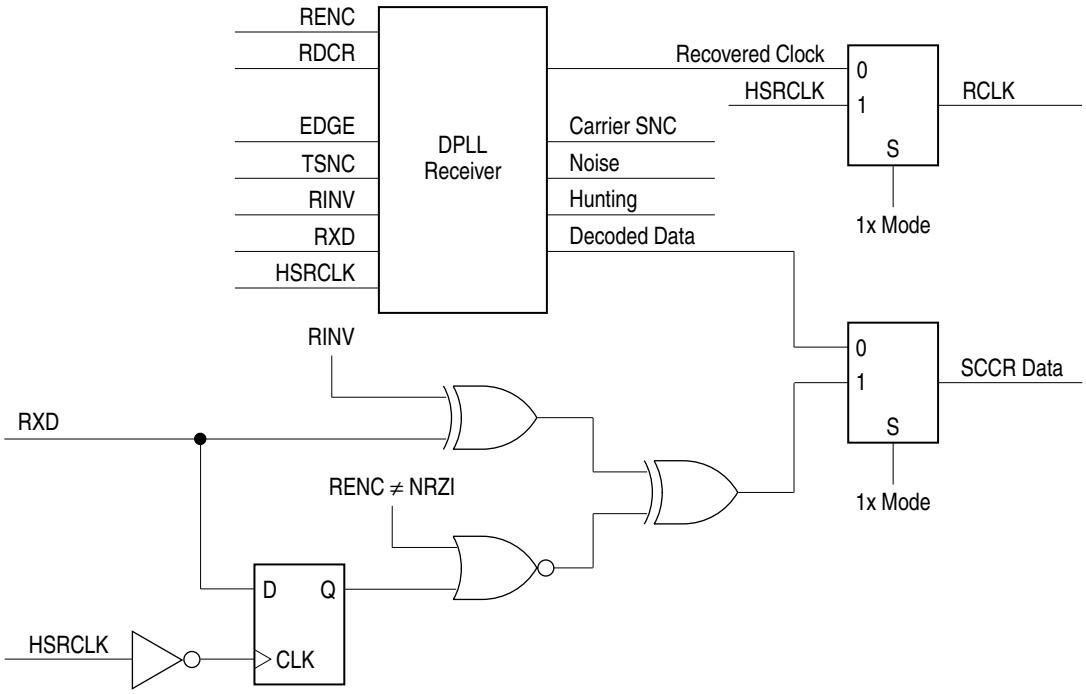


Figure 21-13. DPLL Receiver Block Diagram

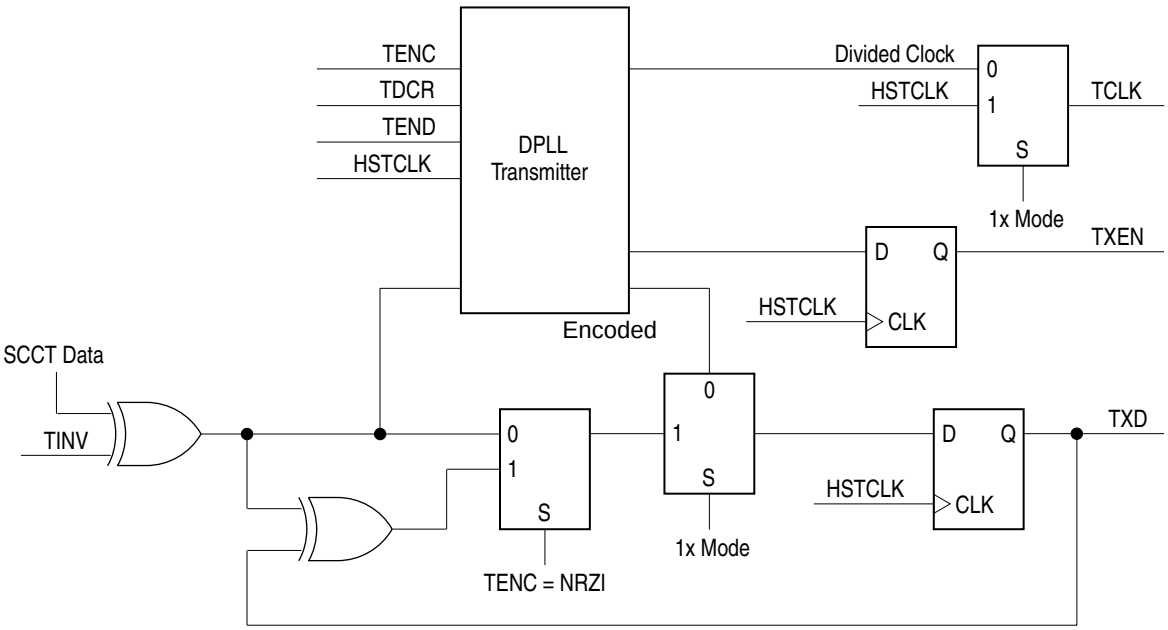


Figure 21-14. DPLL Transmitter Block Diagram

The DPLL can be driven by one of the baud rate generator outputs or an external clock, CLK*x*. In the block diagrams, this clock is labeled HSRCLK/HSTCLK. The HSRCLK/HSTCLK should be approximately 8x, 16x, or 32x the data rate, depending on the coding chosen. The DPLL uses this clock, along with the data stream, to construct a data clock that can be used as the SCC Rx and/or Tx clock. In all modes, the DPLL uses the input clock to determine the nominal bit time. If the DPLL is bypassed, HSRCLK/HSTCLK is used directly as RCLK/TCLK.

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 data stream for transitions; when one is detected, the DPLL adjusts the count to produce an output clock that tracks incoming bits.

The DPLL has a carrier-sense signal that indicates when data transfers are on RXD. The carrier-sense signal asserts as soon as a transition is detected on RXD; it negates after the programmed number of clocks in GSMR_L[TSNC] when no transitions are detected.

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 can function as a preamble; others use the patterns in Table 21-7. When transmission occurs, the SCC can generate preamble patterns, as programmed in GSMR_L[TPP, TPL].

Table 21-7. Preamble Requirements

Decoding Method	Preamble Pattern	Minimum 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	101010...10	8-bit
Differential Manchester	All ones	8-bit

The DPLL can also be used to invert the data stream of a transfer. This feature is available in all encodings, including standard NRZ format. Also, when the transmitter is idling, the DPLL can either force TXD high or continue encoding the data supplied to it.

The DPLL is used for UART encoding/decoding, which gives the option of selecting the divide ratio in the UART decoding process (8×, 16×, or 32×). Typically, 16× is used.

The maximum data rate supported with the DPLL is 3.125 MHz, assuming a 25-MHz system clock and the 8× ratio (25 MHz/8 = 3.125 MHz). Thus, the frequency applied to CLK*x* or generated by an internal baud rate generator may be up to 25 MHz on a 25-MHz MPC855T, if the DPLL 8×, 16×, or 32× option is used.

Note the 1:2 system clock/serial clock ratio does not apply when the DPLL is used to recover the clock in the 8×, 16×, or 32× modes. Synchronization occurs internally after the DPLL generates the Rx clock. Therefore, even the fastest DPLL clock generation (the 8× option) easily meets the required 1:2 ratio clocking limit.

21.4.5.1 Encoding Data with a DPLL

The SCC contains a DPLL unit that can be programmed to encode and decode the SCC data as NRZ, NRZI Mark, NRZI Space, FM0, FM1, Manchester, and Differential Manchester. Figure 21-15 shows the different encoding methods.

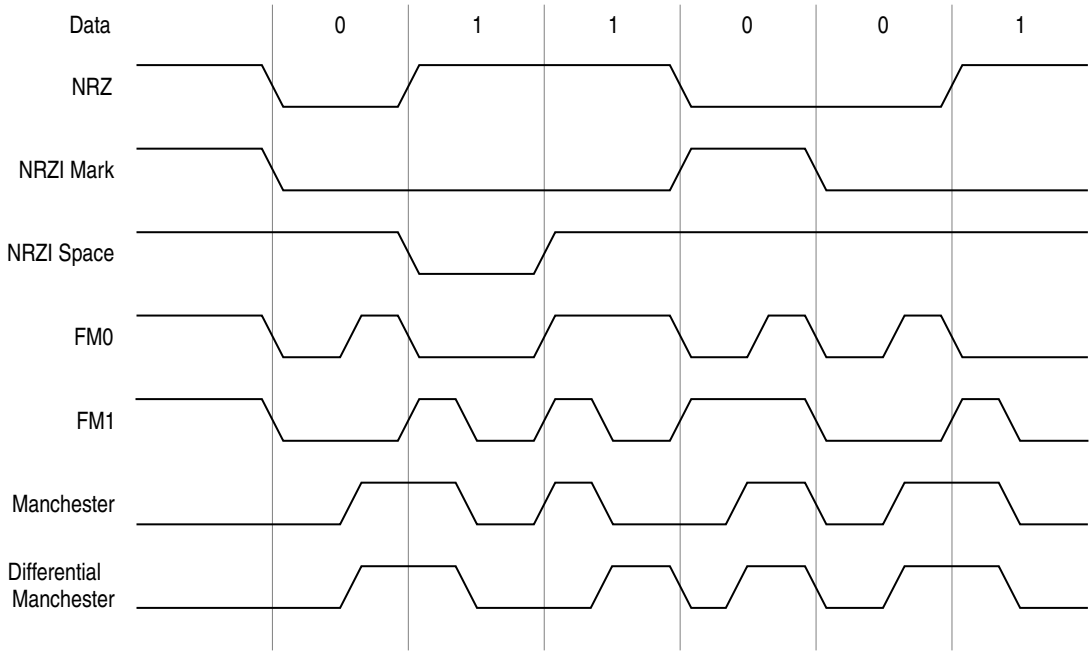


Figure 21-15. DPLL Encoding Examples

If the DPLL is not needed, NRZ or NRZI codings can be selected in GSMR_L[RENC, TENC]. Coding definitions are shown in Table 21-8.

Table 21-8. DPLL Codings

Coding	Description
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.

21.4.6 Clock Glitch Detection

Clock glitches cause problems for many communications systems, and they may go undetected by the system. 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. A clock glitch occurs when more than one edge occurs in a time period that violates the minimum high or low time specification of the input clock.

The SCC on the MPC855T has a special circuit designed to detect glitches and alert the system of a problem at the physical layer. The glitch-detect circuit is not a specification test; if a circuit does not meet the SCC's input clocking specifications, erroneous data may not be detected or false glitch indications can occur. 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 GSMR_H[GDE].

If a spike is detected, a maskable Rx or Tx glitched clock interrupt is generated in SCCE[GLR,GLT]. Although the receiver or transmitter can be reset or allowed to continue operation, statistics on clock glitches should be kept for evaluation to help in debugging, especially during prototype testing.

21.4.7 Reconfiguring the SCC

The proper reconfiguration sequence must be followed for SCC parameters that cannot be changed dynamically. For instance, the internal baud rate generators allow on-the-fly changes, but the DPLL-related GSMR does not. The steps in the following sections show how to disable, reconfigure and re-enable the SCC to ensure that buffers currently in use are properly closed before reconfiguring the SCC and that subsequent data goes to or from new buffers according to the new configuration.

Modifying parameter RAM does not require the SCC to be fully disabled. See the parameter RAM description for when values can be changed. To disable the SCC, SMCs, SPI, and the I^2C, set CPCR[RST] to reset the entire CPM.

21.4.7.1 General Reconfiguration Sequence for the SCC Transmitter

The SCC transmitter can be reconfigured by following these general steps:

1. If the SCC is sending data, issue a STOP TRANSMIT command. Transmission should stop smoothly. If the SCC is not transmitting (no TxBDs are ready or the GRACEFUL STOP TRANSMIT command has been issued and completed) or the INIT TX PARAMETERS command is issued, the STOP TRANSMIT command is not required.
2. Clear GSMR_L[ENT] to disable the SCC transmitter and put it in reset state.
3. Modify SCC Tx parameters or parameter RAM. To switch protocols or restore the initial Tx parameters, issue an INIT TX PARAMETERS command.

4. If an INIT TX PARAMETERS command was not issued in step 3, issue a RESTART TRANSMIT command.
5. Set GSMR_L[ENT]. Transmission begins using the TxBD pointed to by TBPTR, assuming the R bit is set.

21.4.7.2 Reset Sequence for the SCC Transmitter

The following steps reinitialize the SCC transmit parameters to the reset state:

1. Clear GSMR_L[ENT].
2. Make any modifications then issue the INIT TX PARAMETERS command.
3. Set GSMR_L[ENT].

21.4.7.3 General Reconfiguration Sequence for the SCC Receiver

The SCC receiver can be reconfigured by following these steps:

1. Clear GSMR_L[ENR]. The SCC receiver is now disabled and put in a reset state.
2. Modify SCC Rx parameters or parameter RAM. To switch protocols or restore Rx parameters to their initial state, issue an INIT RX PARAMETERS command.
3. If the INIT RX PARAMETERS command was not issued in step 2, issue an ENTER HUNT MODE command.
4. Set GSMR_L[ENR]. Reception begins using the RxBD pointed to by RBPTR, assuming the E bit is set.

21.4.7.4 Reset Sequence for the SCC Receiver

To reinitialize the SCC receiver to the state it was in after reset, follow these steps:

1. Clear GSMR_L[ENR].
2. Make any modifications then issue the INIT RX PARAMETERS command.
3. Set GSMR_L[ENR].

21.4.7.5 Switching Protocols

To switch the SCC’s protocol without resetting the board, follow these steps:

1. Clear GSMR_L[ENT, ENR].
2. Make protocol changes in the GSMR and additional parameters then issue the INIT TX and RX PARAMETERS command to initialize both Tx and Rx parameters.
3. Set GSMR_L[ENT, ENR] to enable the SCC with the new protocol.

21.4.8 Saving Power

To save power when not in use, the SCC can be disabled by clearing GSMR_L[ENT, ENR].

Chapter 22
SCC UART Mode

The universal asynchronous receiver transmitter (UART) protocol is commonly used to send low-speed data between devices. The term asynchronous is used because it is not necessary to send clocking information along with the data being sent. UART links are typically 38400 baud or less and are character-based. Asynchronous links are used to connect terminals with other devices. Even where synchronous communications are required, the UART is often used as a local port to run board debugger software. The character format of the UART protocol is shown in Figure 22-1.

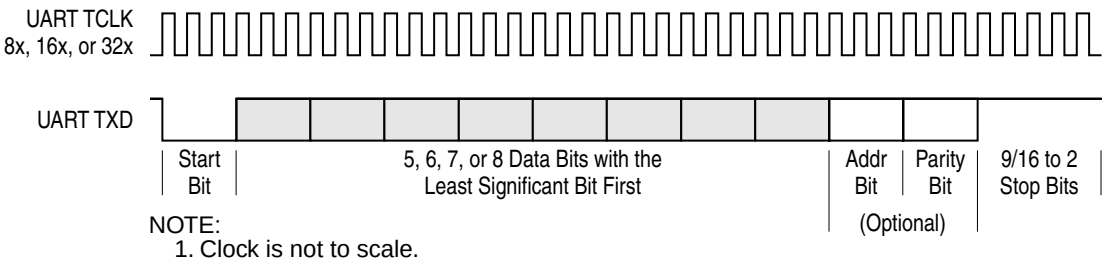


Figure 22-1. UART Character Format

Because the transmitter and receiver operate asynchronously, there is no need to connect the transmit and receive clocks. Instead, the receiver oversamples the incoming data stream (usually by a factor of 16) and uses some of these samples to determine the bit value. Traditionally, the middle 3 of the 16 samples are used. Two UARTs can communicate using this system if the transmitter and receiver use the same parameters, such as the parity scheme and character length.

When data is not sent, a continuous stream of ones is sent (idle condition). Because the start bit is always a zero, the receiver can detect when real data is once again on the line. UART specifies an all-zeros break character, which ends a character transfer sequence.

The most popular protocol that uses asynchronous characters is the RS-232 standard, which specifies baud rates, handshaking protocols, and mechanical/electrical details. Another popular format is RS-485, which defines a balanced line system allowing longer cables than RS-232 links. Even synchronous protocols like HDLC are sometimes defined to run over asynchronous links. The Profibus standard extends UART protocol to include LAN-oriented features such as token passing.

All standards provide handshaking signals, but some systems require only three physical lines—Tx data, Rx data, and ground. Many proprietary standards have been built around the UART's asynchronous character frame, some of which implement a multidrop configuration where multiple stations, each with a specific address, can be present on a network. In multidrop mode, frames of characters are broadcast with the first character acting as a destination address. To accommodate this, the UART frame is extended one bit to distinguish address characters from normal data characters.

In synchronous UART (isochronous operation), a separate clock signal is explicitly provided with the data. Start and stop bits are present in synchronous UART, but oversampling is not required because the clock is provided with each bit.

The general SCC mode register (GSMR) is used to configure an SCC channel to function in UART mode, which provides standard serial I/O using asynchronous character-based (start-stop) protocols with RS-232C-type lines. Using standard asynchronous bit rates and protocols, an SCC UART controller can communicate with any existing RS-232-type device and provides a serial communications port to other microprocessors and terminals (either locally or via modems). The independent transmit and receive sections, whose operations are asynchronous with the core, send data from memory (either internal or external) to TXD and receive data from RXD. The UART controller supports a multidrop mode for master/slave operations with wake-up capability on both the idle signal and address bit. It also supports synchronous operation where a clock (internal or external) must be provided with each bit received.

22.1 Features

The following list summarizes main features of an SCC UART controller:

- Flexible message-based data structure
- Implements synchronous and asynchronous UART
- Multidrop operation
- Receiver wake-up on idle line or address bit
- Receive entire messages into buffers as indicated by receiver idle timeout or by control character reception
- Eight control character comparison
- Two address comparison in multidrop configurations
- Maintenance of four 16-bit error counters
- Received break character length indication
- Programmable data length (5–8 bits)
- Programmable fractional stop bit lengths (from 9/16 to 2 bits) in transmission
- Capable of reception without a stop bit
- Even/odd/force/no parity generation and check

- Frame error, noise error, break, and idle detection
- Transmit preamble and break sequences
- Freeze transmission option with low-latency stop

22.2 Normal Asynchronous Mode

In normal asynchronous mode, the receive shift register receives incoming data on RXD. Control bits in the UART mode register (PSMR) define the length and format of the UART character. Bits are 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; if all do not agree, the noise indication counter (NOSEC) in parameter RAM is incremented. When a complete character has been clocked in, the contents of the receive shift register are transferred to the receive FIFO before proceeding to the receive buffer. The CPM flags UART events, including reception errors, in SCCE and the RxBD status and control fields.

The SCC 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 sends outgoing data on TXD. Data is then clocked synchronously with the transmit clock, which may have either an internal or external source. Characters are sent lsb first. Only the data portion of the UART frame is stored in the buffers because start and stop bits are generated and stripped by the SCC. A parity bit can be generated in transmission and checked during reception; although it is not stored in the buffer, its value can be inferred from the buffer's reporting mechanism. Similarly, the optional address bit is not stored in the transmit or receive buffer, but is supplied in the BD itself. Parity generation and checking includes the optional address bit. GSMR_H[RFW] must be set for an 8-bit receive FIFO in the UART receiver.

22.3 Synchronous Mode

In synchronous mode, the controller uses a 1× data clock for timing. The receive shift register receives incoming data on RXD synchronous with the clock. The bit length and format of the serial character are defined by the control bits in the PSMR in the same way as in asynchronous mode. When a complete byte has been clocked in, the contents of the receive shift register are transferred to the receive FIFO before proceeding to the receive buffer. The CPM flags UART events, including reception errors, in SCCE and the RxBD status and control fields. GSMR_H[RFW] must be set for an 8-bit receive FIFO.

The synchronous UART transmit shift register sends outgoing data on TXD. Data is then clocked synchronously with the transmit clock, which can have an internal or external source.

22.4 SCC UART Parameter RAM

For UART mode, the protocol-specific area of the SCC parameter RAM is mapped as in Table 22-1.

Table 22-1. UART-Specific SCC Parameter RAM Memory Map

Offset [1]	Name	Width	Description
0x30	—	DWord	Reserved
0x38	**MAX_IDL**	Hword	Maximum idle characters. When a character is received, the receiver begins counting idle characters. If MAX_IDL idle characters are received before the next data character, an idle timeout occurs and the buffer is closed, generating a maskable interrupt request to the core to receive the data from the buffer. Thus, MAX_IDL offers a way to demarcate frames. To disable the feature, clear MAX_IDL. The bit length of an idle character is calculated as follows: 1 + data length (5–9) + 1 (if parity is used) + number of stop bits (1–2). For 8 data bits, no parity, and 1 stop bit, the character length is 10 bits.
0x3A	IDLC	Hword	Temporary idle counter. Holds the current idle count for the idle timeout process. IDLC is a down-counter and does not need to be initialized or accessed.
0x3C	**BRKCR**	Hword	Break count register (transmit). Determines the number of break characters the transmitter sends. The transmitter sends a break character sequence when a STOP TRANSMIT command is issued. For 8 data bits, no parity, 1 stop bit, and 1 start bit, each break character consists of 10 zero bits.
0x3E	**PAREC**	Hword	User-initialized,16-bit (modulo–2^{16}) counters incremented by the CP. PAREC counts received parity errors. FRMEC counts received characters with framing errors. NOSEC counts received characters with noise errors. BRKEC counts break conditions on the signal. A break condition can last for hundreds of bit times, yet BRKEC is incremented only once during that period.
0x40	**FRMEC**	Hword	
0x42	**NOSEC**	Hword	
0x44	**BRKEC**	Hword	
0x46	BRKLN	Hword	Last received break length. Holds the length of the last received break character sequence measured in character units. For example, if RXD is low for 20 bit times and the defined character length is 10 bits, BRKLN = 0x002, indicating that the break sequence is at least 2 characters long. BRKLN is accurate to within one character length.
0x48	**UADDR1**	Hword	UART address character 1/2. In multidrop mode, the receiver provides automatic address recognition for two addresses. In this case, program the lower order bytes of UADDR1 and UADDR2 with the two preferred addresses.
0x4A	**UADDR2**	Hword	
0x4C	RTEMP	Hword	Temporary storage
0x4E	**TOSEQ**	Hword	Transmit out-of-sequence character. Inserts out-of-sequence characters, such as XOFF and XON, into the transmit stream. The TOSEQ character is put in the Tx FIFO without affecting a Tx buffer in progress. See Section 22.11, “Inserting Control Characters into the Transmit Data Stream.”

Table 22-1. UART-Specific SCC Parameter RAM Memory Map (continued)

0x50	**CHARACTER1**	Hword	Control character 1–8. These characters define the Rx control characters on which interrupts can be generated.
0x52	**CHARACTER2**	Hword	
0x54	**CHARACTER3**	Hword	
0x56	**CHARACTER4**	Hword	
0x58	**CHARACTER5**	Hword	
0x5A	**CHARACTER6**	Hword	
0x5C	**CHARACTER7**	Hword	
0x5E	**CHARACTER8**	Hword	
0x60	**RCCM**	Hword	Receive control character mask. Used to mask comparison of CHARACTER1–8 so classes of control characters can be defined. A one enables the comparison, and a zero masks it.
0x62	RCCR	Hword	Receive control character register. Used to hold the last rejected control character (not written to the Rx buffer). Generates a maskable interrupt. If the core does not process the interrupt and read RCCR before a new control character arrives, the previous control character is overwritten.
0x64	RLBC	Hword	Receive last break character. Used in synchronous UART when PSMR[RZS] = 1; holds the last break character pattern. By counting zeros in RLBC, the core can measure break length to a one-bit resolution. Read RLBC by counting the zeros written from bit 0 to where the first one was written. RLBC = 0b001xxxxxxxxxxxx indicates two zeros; 0b1xxxxxxxxxxxxxx indicates no zeros. Note that RLBC can be used in combination with BRKLN above to measure the break length down to a bit resolution: (BRKLN + number of zeros in RLBC).

1 From SCC base. SCC base = IMMR + 0x3C00 (SCC1)

22.5 Data-Handling Methods: Character- or Message-Based

An SCC UART controller uses the same BD table and buffer structures as the other protocols and supports both multibuffer, message-based and single-buffer, character-based operation.

For character-based transfers, each character is sent with stop bits and parity and received into separate 1-byte buffers. A maskable interrupt is generated when each buffer is received.

In a message-based environment, transfers can be made on entire messages rather than on individual characters. To simplify programming and save processor overhead, a message is transferred as a linked list of buffers without core intervention. For example, before handling input data, a terminal driver may wait for an end-of-line character or an idle timeout rather than be interrupted when each character is received. Conversely, ASCII files can be sent as messages ending with an end-of-line character.

When receiving messages, up to eight control characters can be configured to mark the end of a message or generate a maskable interrupt without being stored in the buffer. This option

is useful when flow control characters such as XON or XOFF are needed but are not part of the received message. See Section 22.9, “Receiving Control Characters.”

22.6 Error and Status Reporting

Overrun, parity, noise, and framing errors are reported via the BDs and/or error counters in the UART parameter RAM. Signal status is indicated in the status register; a maskable interrupt is generated when status changes.

22.7 SCC UART Commands

The transmit commands in Table 22-2 are issued to the CP command register (CPCR).

Table 22-2. Transmit Commands

Command	Description
STOP TRANSMIT	After a hardware or software reset and a channel is enabled in the GSMR, the transmitter starts polling the first BD in the TxBD table every 8 Tx clocks. STOP TRANSMIT disables character transmission. If the SCC receives STOP TRANSMIT as a message is being sent, the message is aborted. The transmitter finishes sending data transferred to its FIFO and stops. The TBPTR is not advanced. The UART transmitter sends a programmable break sequence and starts sending idles. The number of break characters in the sequence (which can be zero) should be written to BRKCR in the parameter RAM before issuing this command.
GRACEFUL STOP TRANSMIT	Used to stop transmitting smoothly. The transmitter stops after the current buffer has been completely sent or immediately if no buffer is being sent. SCCE[GRA] is set once transmission stops, then the UART Tx parameters, including the TxBD, can be modified. TBPTR points to the next TxBD in the table. Transmission begins once the R bit of the next BD is set and a RESTART TRANSMIT command is issued.
RESTART TRANSMIT	Enables transmission. The controller expects this command after it disables the channel in its PSMR, after a STOP TRANSMIT command, after a GRACEFUL STOP TRANSMIT command, or after a transmitter error. Transmission resumes from the current BD.
INIT TX PARAMETERS	Resets the transmit parameters in the parameter RAM. Issue only when the transmitter is disabled. Note that INIT TX AND RX PARAMETERS resets both Tx and Rx parameters.

Receive commands are described in Table 22-3.

Table 22-3. Receive Commands

Command	Description
ENTER HUNT MODE	Forces the receiver to close the RxBD in use and enter hunt mode. After a hardware or software reset, once an SCC is enabled in the GSMR, the receiver is automatically enabled and uses the first BD in the RxBD table. If a message is in progress, the receiver continues receiving in the next BD. In multidrop hunt mode, the receiver continually scans the input data stream for the address character. When it is not in multidrop mode, it waits for the idle sequence (one character of idle). Data present in the Rx FIFO is not lost when this command is executed.
CLOSE RXBD	Forces the SCC to close the RxBD in use and use the next BD for subsequent received data. If the SCC is not in the process of receiving data, no action is taken. Note that in an SCC UART controller, CLOSE RXBD functions like ENTER HUNT MODE but does not need to receive an idle character to continue receiving.
INIT RX PARAMETERS	Resets the receive parameters in the parameter RAM. Should be issued when the receiver is disabled. Note that INIT TX AND RX PARAMETERS resets both Tx and Rx parameters.

22.8 Multidrop Systems and Address Recognition

In multidrop systems, more than two stations can be on a network, each with a specific address. Figure 22-2 shows two examples of this configuration. Frames made up of many characters can be broadcast as long as the first character is the destination address. The UART frame is extended by one bit to distinguish an address character from standard data characters. Programmed in PSMR[UM], the controller supports the following two multidrop modes:

- Automatic multidrop mode—The controller checks the incoming address character and accepts subsequent data only if the address matches one of two user-defined values. The two 16-bit address registers, UADDR1 and UADDR2, support address recognition. Only the lower 8 bits are used so the upper 8 bits should be cleared; for addresses less than 8 bits, unused high-order bits should also be cleared. The incoming address is checked against UADDR1 and UADDR2. When a match occurs, RxBD[AM] indicates whether UADDR1 or UADDR2 matched.
- Manual multidrop mode—The controller receives all characters. An address character is always written to a new buffer and can be followed by data characters. User software performs the address comparison.

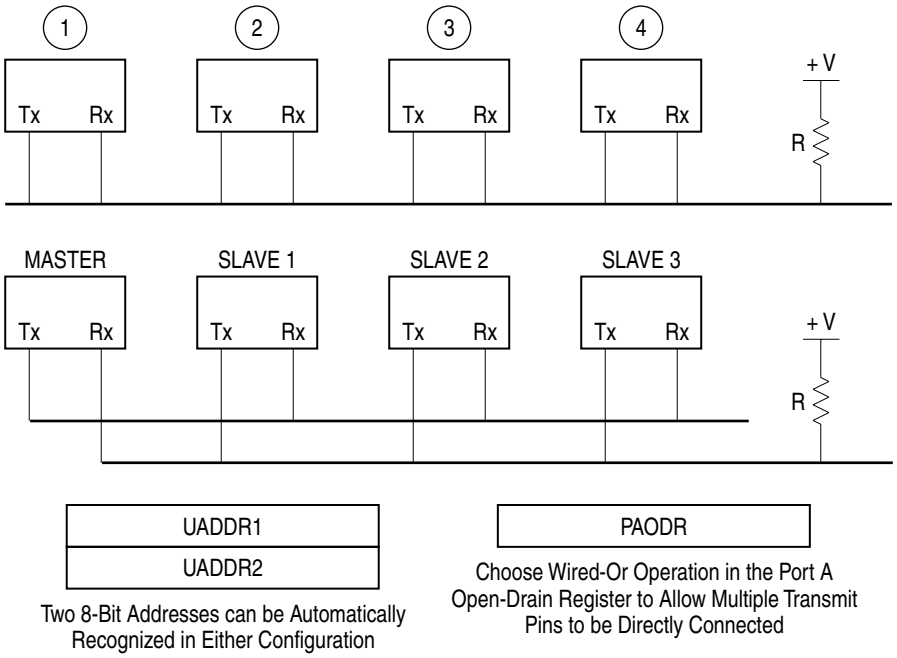


Figure 22-2. Two UART Multidrop Configurations

22.9 Receiving Control Characters

The UART receiver can recognize special control characters used in a message-based environment. Eight control characters can be defined in a control character table in the UART parameter RAM. Each incoming character is compared to the table entries using a

mask (the received control character mask, RCCM) to strip don't cares. If a match occurs, the received control character can either be written to the receive buffer or rejected.

If the received control character is not rejected, it is written to the receive buffer. The receive buffer is then automatically closed to allow software to handle end-of-message characters. Control characters that are not part of the actual message, such as XOFF, can be rejected. Rejected characters bypass the receive buffer and are written directly to the received control character register (RCCR), which triggers maskable interrupt.

The 16-bit entries in the control character table support control character recognition. Each entry consists of the control character, a valid bit (end of table), and a reject bit. See Figure 22-3.

Offset[1]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x50	**E**	**R**	—						**CHARACTER1**							
0x52	**E**	**R**	—						**CHARACTER2**							
⋮	⋮	⋮	⋮						⋮							
0x5E	**E**	**R**	—						**CHARACTER8**							
0x60	**1**	**1**	—						**RCCM**							
0x62	—								RCCR							

[1] From SCC*x* base address

Figure 22-3. Control Character Table, RCCM, and RCCR

Table 22-4 describes the data structure used in control character recognition.

Table 22-4. Control Character Table, RCCM, and RCCR Descriptions

Offset	Bits	Name	Description
0x50–0x5E	0	**E**	End of table. In tables with eight control characters, E is always 0. 0 This entry is valid. 1 The entry is not valid and is not used.
	1	**R**	Reject character. 0 A matching character is not rejected but is written into the Rx buffer, which is then closed. If RxBD[I] is set, the buffer closing generates a maskable interrupt through SCCE[RX]. A new buffer is opened if more data is in the message. 1 A matching character is written to RCCR and not to the Rx buffer. A maskable interrupt is generated through SCCE[CCR]. The current Rx buffer is not closed.
	2–7	—	Reserved
	8–15	**CHARACTER*n***	Control character values 1–8. Defines control characters to be compared to the incoming character. For characters smaller than 8 bits, the most significant bits should be zero.

Table 22-4. Control Character Table, RCCM, and RCCR Descriptions (continued)

Offset	Bits	Name	Description
0x60	0–1	**0b11**	Must be set. Used to mark the end of the control character table in case eight characters are used. Setting these bits ensures correct operation during control character recognition.
	2–7	—	Reserved
	8–15	**RCCM**	Received control character mask. Used to mask the comparison of CHARACTER*n*. Each RCCM bit corresponds to the respective bit of CHARACTER*n* and decodes as follows. 0 Ignore this bit when comparing the incoming character to CHARACTER*n*. 1 Use this bit when comparing the incoming character to CHARACTER*n*.
0x62	0–7	—	Reserved
	8–15	RCCR	Received control character register. If the newly arrived character matches and is rejected from the buffer (R = 1), the PIP controller writes the character into the RCCR and generates a maskable interrupt. If the core does not process the interrupt and read RCCR before a new control character arrives, the previous control character is overwritten.

22.10 Hunt Mode (Receiver)

A UART receiver in hunt mode remains deactivated until an idle or address character is recognized, depending on PSMR[UM]. A receiver is forced into hunt mode by issuing an ENTER HUNT MODE command.

The receiver aborts any message in progress when ENTER HUNT MODE is issued. When the message is finished, the receiver is reenabled by detecting the idle line (one idle character) or by the address bit of the next message, depending on PSMR[UM]. When a receiver in hunt mode receives a break sequence, it increments BRKEC and generates a BRK interrupt condition.

22.11 Inserting Control Characters into the Transmit Data Stream

The SCC UART transmitter can send out-of-sequence, flow-control characters like XON and XOFF. The controller polls the transmit out-of-sequence register (TOSEQ), shown in Figure 22-4, 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. The TOSEQ character (in CHARSEND) is sent 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 sent for eight (SCC1) character times. To reduce this latency, set GSMR_H[TFL] to decrease the FIFO size to one character before enabling the transmitter.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—		REA	I	CT	—		A	CHARSEND							
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	SCC base + 0x4E															

Figure 22-4. Transmit Out-of-Sequence Register (TOSEQ)

Table 22-5 describes TOSEQ fields.

Table 22-5. TOSEQ Field Descriptions

Bit	Name	Description
0–1	—	Reserved, should be cleared.
2	**REA**	Ready. Set when the character is ready for transmission. Remains 1 while the character is being sent. The CP clears this bit after transmission.
3	**I**	Interrupt. If this bit is set, transmission completion is flagged in the event register (SCCE[TX] is set), triggering a maskable interrupt to the core.
4	CT	Clear-to-send lost. Operates only if the SCC monitors $\overline{\text{CTS}}$ (GSMR_L[DIAG]). The CP sets this bit if $\overline{\text{CTS}}$ negates when the TOSEQ character is sent. If $\overline{\text{CTS}}$ negates and the TOSEQ character is sent during a buffer transmission, the TxBD[CT] status bit is also set.
5–6	—	Reserved, should be cleared.
7	**A**	Address. Setting this bit indicates an address character for multidrop mode.
8–15	**CHARSEND**	Character send. Contains the character to be sent. Any 5- to 8-bit character value can be sent in accordance with the UART configuration. The character should be placed in the lsbs of CHARSEND. This value can be changed only while REA = 0.

22.12 Sending a Break (Transmitter)

A break is an all-zeros character with no stop bit that is sent by issuing a STOP TRANSMIT command. The SCC finishes transmitting outstanding data, sends a programmable number of break characters (determined by BRKCR), and reverts to idle or sends data if a RESTART TRANSMIT command is given before completion. When the break code is complete, the transmitter sends at least one high bit before sending more data, to guarantee recognition of a valid start bit. Because break characters do not preempt characters in the transmit FIFO, they may not be sent for eight (SCC1) character times. To reduce this latency, set GSMR_H[TFL] to decrease the FIFO size to one character before enabling the transmitter.

22.13 Sending a Preamble (Transmitter)

Sending a preamble sequence of consecutive ones ensures that a line is idle before sending a message. If the preamble bit TxBD[P] is set, the SCC sends a preamble sequence (idle character) before sending the 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.

22.14 Fractional Stop Bits (Transmitter)

The asynchronous UART transmitter can be programmed to send fractional stop bits. The FSB field in the data synchronization register (DSR) determines the fractional length of the last stop bit to be sent. FSB can be modified at any time. If two stop bits are sent, only the second is affected. Idle characters are always sent as full-length characters.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—	FSB				—	—	—	—	—	—	—	—	—	—	—
Reset	0	1				1	1	0	0	1	1	1	1	1	1	0
R/W	R/W															
Addr	0xA0E (DSR1)															

Figure 22-5. Data Synchronization Register (DSR)

Table 22-6 describes DSR fields.

Table 22-6. DSR Fields Descriptions

Bit	Name	Description
0	—	0b0
1–4	FSB	Fractional stop bits. For 16× oversampling: 1111 Last transmitted stop bit 16/16. Default value after reset. 1110 Last transmitted stop bit 15/16. … 1000 Last transmitted stop bit 9/16. 0xxx Invalid. Do not use. For 32× oversampling: 1111 Last transmitted stop bit 32/32. Default value after reset. 1110 Last transmitted stop bit 31/32. … 0000 Last transmitted stop bit 17/32. For 8× oversampling: 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. 0xxx Invalid. Do not use. The UART receiver can always receive fractional stop bits. The next character's start bit can begin any time after the three middle samples have been taken.
5–6	—	0b11
7–8	—	0b00
9–14	—	0b111111
15	—	0b0

22.15 Handling Errors in the SCC UART Controller

The UART controller reports character reception and transmission error conditions via the BDs, the error counters, and the SCCE. Modem interface lines can be monitored by the port C pins. Transmission errors are described in Table 22-7.

Table 22-7. Transmission Errors

Error	Description
$\overline{\text{CTS}}$ Lost during Character Transmission	When $\overline{\text{CTS}}$ negates during transmission, the channel stops after finishing the current character. The CP sets TxBD[CT] and generates the TX interrupt if it is not masked. The channel resumes transmission after the RESTART TRANSMIT command is issued and $\overline{\text{CTS}}$ is asserted. Note that if $\overline{\text{CTS}}$ is used, the UART also offers an asynchronous flow control option that does not generate an error. See the description of PSMR[FLC] in Table 22-9.

Reception errors are described in Table 22-8.

Table 22-8. Reception Errors

Error	Description
Overrun	Occurs when the channel overwrites the previous character in the Rx FIFO with a new character, losing the previous character. The channel then writes the new character to the buffer, closes it, sets RxBD[OV], and generates an RX interrupt if not masked. In automatic multidrop mode, the receiver enters hunt mode immediately.
$\overline{\text{CD}}$ Lost during Character Reception	If this error occurs and the channel is using this pin to automatically control reception, the channel terminates character reception, closes the buffer, sets RxBD[CD], and generates the RX interrupt if not masked. 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	When a parity error occurs, the channel writes the received character to the buffer, closes the buffer, sets RxBD[PR], and generates the RX interrupt if not masked. The channel also increments the parity error counter PAREC. In automatic multidrop mode, the receiver enters hunt mode immediately.
Noise	A noise error occurs when the three samples of a bit are not identical. When this error occurs, the channel writes the received character to the buffer, proceeds normally, but increments the noise error counter NOSEC. Note that this error does not occur in synchronous mode.
Idle Sequence Receive	If the UART is receiving data and gets an idle character (all ones), the channel begins counting consecutive idle characters received. If MAX_IDL is reached, the buffer is closed and an RX interrupt is generated if not masked. If no 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. To disable the idle sequence function, clear MAX_IDL.

Table 22-8. Reception Errors (continued)

Error	Description
Framing	The UART reports a framing errors when it receives a character with no stop bit, regardless of the mode. The channel writes the received character to the buffer, closes it, sets RxBD[FR], generates the RX interrupt if not masked, increments FRMEC, but does not check parity for this character. In automatic multidrop mode, the receiver immediately enters hunt mode. If the UART allows data with no stop bits (PSMR[RZS] = 1) when in synchronous mode (PSMR[SYN] = 1), framing errors are reported but reception continues assuming the unexpected zero is the start bit of the next character; in this case, the user may ignore a reported framing error until multiple framing errors occur within a short period.
Break Sequence	When the first break sequence is received, the UART increments the break error counter BRKEC. It updates BRKLN when the sequence completes. After the first 1 is received, the UART sets SCCE[BRKE], which generates an interrupt if not masked. If the UART is receiving characters when it receives a break, it closes the Rx buffer, sets RxBD[BR], and sets SCCE[RX], which can generate an interrupt if not masked. If PSMR[RZS] = 1 when the UART is in synchronous mode, a break sequence is detected after two successive break characters are received.

22.16 UART Mode Register (PSMR)

For UART mode, the SCC protocol-specific mode register (PSMR) is called the UART mode register. Many bits can be modified while the receiver and transmitter are enabled. Figure 22-6 shows the PSMR in UART mode.

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	—	PEN	RPM		TPM	
Reset	0															
R/W	R/W															
Addr	0xA08 (PSMR1)															

Figure 22-6. Protocol-Specific Mode Register for UART (PSMR)

Table 22-9 describes PSMR UART fields.

Table 22-9. PSMR UART Field Descriptions

Bit	Name	Description
0	FLC	Flow control. 0 Normal operation. The GSMR and port C registers determine the mode of $\overline{\text{CTS}}$. 1 Asynchronous flow control. When $\overline{\text{CTS}}$ 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 is sent before transmission stops. When $\overline{\text{CTS}}$ is asserted again, transmission continues where it left off and no $\overline{\text{CTS}}$ lost error is reported. Only idle characters are sent while $\overline{\text{CTS}}$ is negated.
1	SL	Stop length. Selects the number of stop bits the SCC sends. SL can be modified on-the-fly. The receiver is always enabled for one stop bit unless the SCC UART is in synchronous mode and PSMR[RZS] is set. Fractional stop bits are configured in the DSR. 0 One stop bit. 1 Two stop bits.

Table 22-9. PSMR UART Field Descriptions (continued)

Bit	Name	Description
2–3	CL	Character length. Determines the number of data bits in the character, not including optional parity or multidrop address bits. If a character is less than 8 bits, most-significant bits are received as zeros and are ignored when the character is sent. CL can be modified on-the-fly. 00 5 data bits 01 6 data bits 10 7 data bits 11 8 data bits
4–5	UM	UART mode. Selects the asynchronous channel protocol. UM can be modified on-the-fly. 00 Normal UART operation. Multidrop mode is disabled and idle-line wake-up mode is selected. The UART receiver leaves hunt mode by receiving an idle character (all ones). 01 Manual multidrop mode. An additional address/data bit is sent with each character. Multidrop asynchronous modes are compatible with the MC68681 DUART, MC68HC11 SCI, DSP56000 SCI, and Intel 8051 serial interface. The receiver leaves hunt mode when the address/data bit is a one, indicating the received character is an address that all inactive processors must process. The 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 subsequent characters. 10 Reserved. 11 Automatic multidrop mode. The CPM compares the address of an incoming address character with UADDR*x* parameter RAM values; subsequent data is accepted only if a match occurs.
6	FRZ	Freeze transmission. Allows the UART transmitter to pause and later continue from that point. 0 Normal operation. If the buffer was previously frozen, it resumes transmission from the next character in the same buffer that was frozen. 1 The SCC completes transmission of any data already transferred to the Tx FIFO (the number of characters depends on GSMR_H[TFL]) and then freezes. After FRZ is cleared, transmission resumes from the next character.
7	RZS	Receive zero stop bits. 0 The receiver operates normally, but at least one stop bit is needed between characters. A framing error is issued if a stop bit is missing. Break status is set if an all-zero character is received with a zero stop bit. 1 Configures the receiver to receive data without stop bits. Useful in V.14 applications where SCC UART controller data is supplied synchronously and all stop bits of a particular character can be omitted for cross-network rate adaptation. RZS should be set only if SYN is set. The receiver continues if a stop bit is missing. 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 reported only after two consecutive break characters have no stop bits.
8	SYN	Synchronous mode. 0 Normal asynchronous operation. GSMR_L[TENC,RENC] must select NRZ and GSMR_L[TDCR, RDCR] select either 8×, 16×, or 32×. 16× is recommended for most applications. 1 Synchronous SCC UART controller using 1× clock (isochronous UART operation). GSMR_L[TENC, RENC] must select NRZ and GSMR_L[RDCR, TDCR] select 1× mode. A bit is transferred with each clock and is synchronous to the clock, which can be internal or external.
9	DRT	Disable receiver while transmitting. 0 Normal operation. 1 While the SCC is sending data, the internal $\overline{\text{RTS}}$ disables and gates the receiver. Useful for a multidrop configuration in which the user does not want to receive its own transmission. For multidrop UART mode, set the BDs' preamble bit, TxBD[P].
10	—	Reserved, should be cleared.

Table 22-9. PSMR UART Field Descriptions (continued)

Bit	Name	Description
11	PEN	Parity enable. 0 No parity. 1 Parity is enabled and determined by the parity mode bits.
12–13, 14–15	RPM, TPM	Receiver/transmitter parity mode. Selects the type of parity check the receiver/transmitter performs; can be modified on-the-fly. Receive parity errors can be ignored but not disabled. 00 Odd parity. If a transmitter counts an even number of ones in the data word, it sets the parity bit so an odd number is sent. If a receiver receives an even number, a parity error is reported. 01 Low parity (space parity). A transmitter sends a zero in the parity bit position. If a receiver does not read a 0 in the parity bit, a parity error is reported. 10 Even parity. Like odd parity, the transmitter adjusts the parity bit, as necessary, to ensure that the receiver receives an even number of one bits; otherwise, a parity error is reported. 11 High parity (mark parity). The transmitter sends a one in the parity bit position. If the receiver does not read a 1 in the parity bit, a parity error is reported.

22.17 SCC UART Receive Buffer Descriptor (RxBD)

The CPM uses RxBDs to report on each buffer received. The CPM closes the current buffer, generates a maskable interrupt, and starts receiving data into the next buffer after one of the following occurs:

- 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.
- An ENTER HUNT MODE or CLOSE RXBD command is issued.
- An address character is received in multidrop mode. The address character is written to the next buffer for a software comparison.

Figure 22-7 shows an example of how RxBDs are used in receiving.

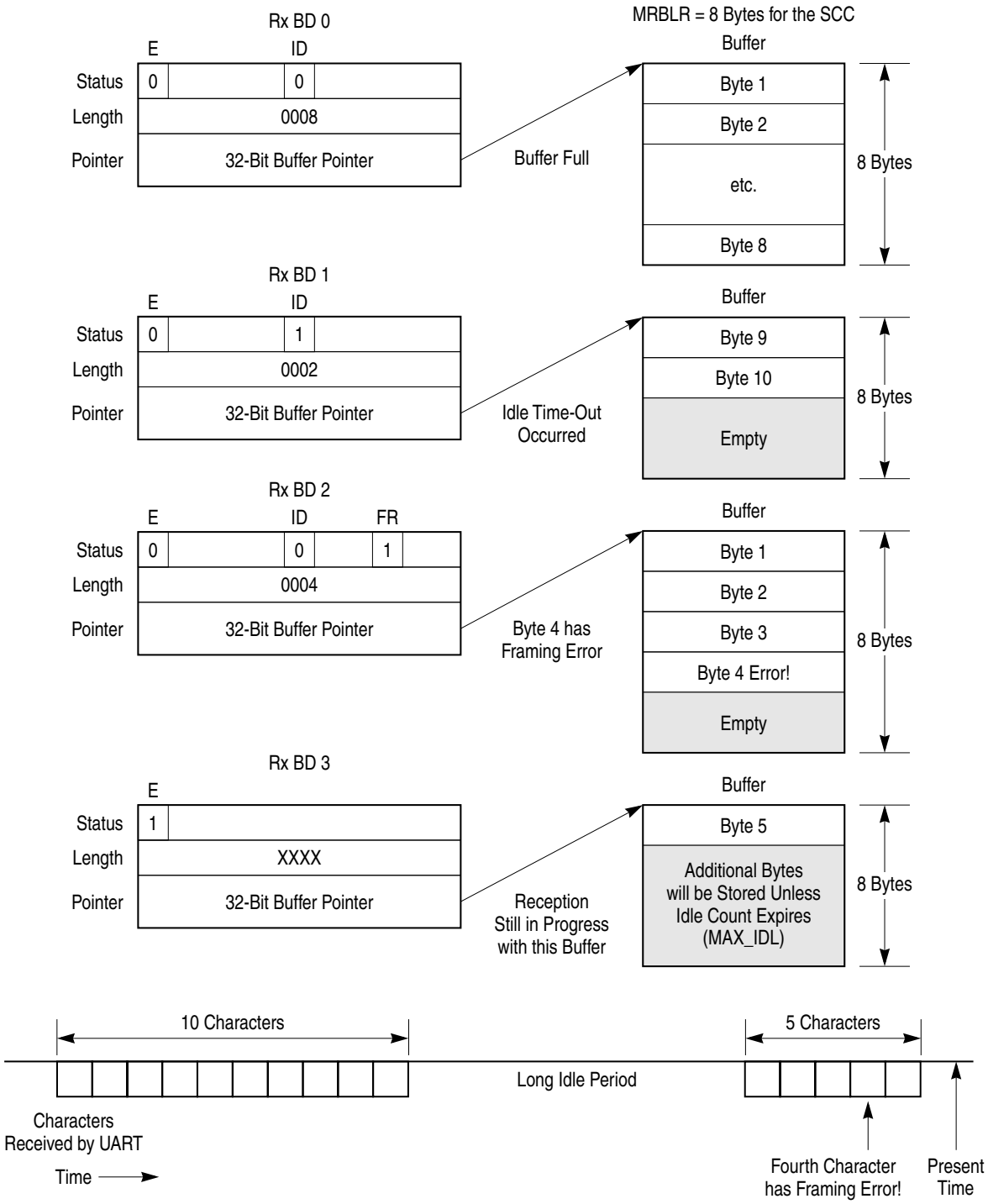


Figure 22-7. SCC UART Receiving using RxBDs

Figure 22-8 shows the SCC UART RxBD.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**E**	—	**W**	**I**	C	A	**CM**	ID	AM	—	BR	FR	PR	—	OV	CD
Offset + 2	Data Length															
Offset + 4	Rx Buffer Pointer															
Offset + 6																

Figure 22-8. SCC UART RxBD

Table 22-10 describes RxBD status and control fields.

Table 22-10. SCC UART RxBD Status and Control Field Descriptions

Bits	Name	Description
0	**E**	Empty. 0 The buffer is full or reception was aborted due to an error. The core can read or write to any fields of this BD. The CPM does not reuse this BD while E = 0. 1 The buffer is not full. The CPM controls this BD and buffer. The core should not modify this BD.
1	—	Reserved, should be cleared.
2	**W**	Wrap (last buffer descriptor in the BD table). 0 Not the last descriptor in the table. 1 Last descriptor in the table. After this buffer is used, the CPM receives incoming data using the BD pointed to by RBASE. The number of BDs in this table is programmable and determined only by the W bit.
3	**I**	Interrupt. 0 No interrupt is generated after this buffer is filled. 1 The CP sets SCCE[RX] when this buffer is completely filled by the CPM, indicating the need for the core to process the buffer. Setting SCCE[RX] causes an interrupt if not masked.
4	C	Control character. 0 This buffer does not contain a control character. 1 The last byte in this buffer matches a user-defined control character.
5	A	Address. 0 The buffer contains only data. 1 For manual multidrop mode, A indicates the first byte of this buffer is an address byte. Software should perform address comparison. In automatic multidrop mode, A indicates the buffer contains a message received immediately after an address matched UADDR1 or UADDR2. The address itself is not written to the buffer but is indicated by the AM bit.
6	**CM**	Continuous mode. 0 Normal operation. The CPM clears E after this BD is closed. 1 The CPM does not clear E after this BD is closed, allowing the buffer to be overwritten when the CPM accesses this BD again. E is cleared if an error occurs during reception, regardless of CM.
7	ID	Buffer closed on reception of idles. The buffer is closed because a programmable number of consecutive idle sequences (MAX_IDL) was received.
8	AM	Address match. Significant only if the address bit is set and automatic multidrop mode is selected in PSMR[UM]. After an address match, AM identifies which user-defined address character was matched. 0 The address matched the value in UADDR2. 1 The address matched the value in UADDR1.
9	—	Reserved, should be cleared.

Table 22-10. SCC UART RxBD Status and Control Field Descriptions (continued)

Bits	Name	Description
10	BR	Break received. Set when a break sequence is received as data is being received into this buffer.
11	FR	Framing error. Set when a character with a framing error (a character without a stop bit) is received and located in the last byte of this buffer. A new Rx buffer is used to receive subsequent data.
12	PR	Parity error. Set when a character with a parity error is received and located in the last byte of this buffer. A new Rx buffer is used to receive subsequent data.
13	—	Reserved, should be cleared.
14	OV	Overrun. Set when a receiver overrun occurs during reception.
15	CD	Carrier detect lost. Set when the carrier detect signal is negated during reception.

Section 21.3, “SCC Buffer Descriptors (BDs),” describes the data length and buffer pointer fields.

22.18 SCC UART Transmit Buffer Descriptor (TxBD)

The CPM uses BDs to confirm transmission and indicate error conditions so the core knows that buffers have been serviced. Figure 22-9 shows the SCC UART TxBD.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**R**	—	**W**	**I**	**CR**	**A**	**CM**	**P**	**NS**	—						CT
Offset + 2	**Data Length**															
Offset + 4	**Tx Buffer Pointer**															
Offset + 6																

Figure 22-9. SCC UART Transmit Buffer Descriptor (TxBD)

Table 22-11 describes TxBD status and control fields.

Table 22-11. SCC UART TxBD Status and Control Field Descriptions

Bit	Name	Description
0	**R**	Ready. 0 The buffer is not ready. This BD and buffer can be modified. The CPM automatically clears R after the buffer is sent or an error occurs. 1 The user-prepared buffer is waiting to begin transmission or is being transmitted. Do not modify the BD once R is set.
1	—	Reserved, should be cleared.
2	**W**	Wrap (last buffer descriptor in TxBD table). 0 Not the last BD in the table. 1 Last BD in the table. After this buffer is used, the CPM sends data using the BD pointed to by TBASE. The number of TxBDs in this table is determined only by the W bit.
3	**I**	Interrupt. 0 No interrupt is generated after this buffer is processed. 1 SCCE[TX] is set after this buffer is processed by the CPM, which can cause an interrupt.

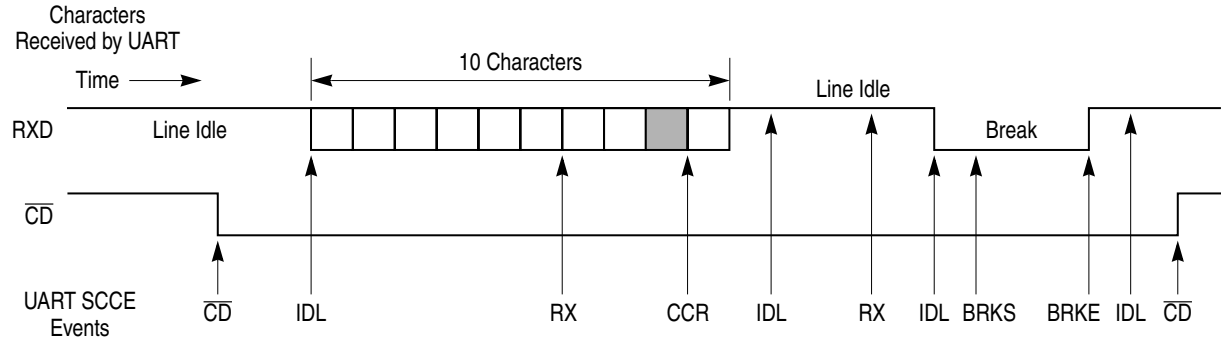
Table 22-11. SCC UART TxBD Status and Control Field Descriptions (continued)

Bit	Name	Description
4	**CR**	Clear-to-send report. 0 The next buffer is sent with no delay (assuming it is ready), but if a $\overline{\text{CTS}}$ lost condition occurs, TxBD[CT] may not be set in the correct TxBD or may not be set at all. Asynchronous flow control, however, continues to function normally. 1 Normal $\overline{\text{CTS}}$ lost error reporting and three bits of idle are sent between consecutive buffers.
5	**A**	Address. Valid only in multidrop mode—automatic or manual. 0 This buffer contains only data. 1 This buffer contains address characters. All data in this buffer is sent as address characters.
6	**CM**	Continuous mode. 0 Normal operation. The CPM clears R after this BD is closed. 1 The CPM does not clear R after this BD is closed, allowing the buffer to be resent next time the CPM accesses this BD. However, R is cleared by transmission errors, regardless of CM.
7	**P**	Preamble. 0 No preamble sequence is sent. 1 Before sending data, the controller sends an idle character consisting of all ones. If the data length of this BD is zero, only a preamble is sent.
8	**NS**	No stop bit or shaved stop bit sent. 0 Normal operation. Stop bits are sent with all characters in this buffer. 1 If PSMR[SYN] = 1, data in this buffer is sent without stop bits. If SYN = 0, the stop bit is shaved, depending on the DSR setting; see Section 22.14, “Fractional Stop Bits (Transmitter).”
9–14	—	Reserved, should be cleared.
15	CT	$\overline{\text{CTS}}$ lost. The CPM writes this status bit after sending the associated buffer. 0 $\overline{\text{CTS}}$ remained asserted during transmission. 1 $\overline{\text{CTS}}$ negated during transmission.

The data length and buffer pointer fields are described in Section 21.3, “SCC Buffer Descriptors (BDs).”

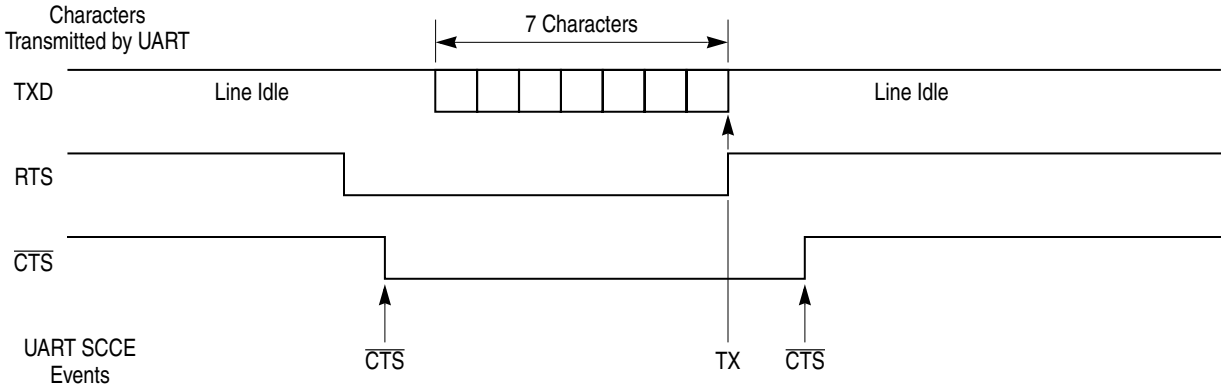
22.19 SCC UART Event Register (SCCE) and Mask Register (SCCM)

The SCC event register (SCCE) is used to report events recognized by the UART channel and to generate interrupts. When an event is recognized, the controller sets the corresponding SCCE bit. Interrupts can be masked in the UART mask register (SCCM), which has the same format as SCCE. Setting a mask bit enables the corresponding SCCE interrupt; clearing a bit masks it. Figure 22-10 shows example interrupts that can be generated by the SCC UART controller.



Notes:
1. The first RX event assumes Rx 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 the SCC itself.

Legend:
A receive control character defined not to be stored in the Rx buffer.



Notes:
1. TX event assumes all seven characters were put into a single buffer and TxBD[CR]=1.
2. The CTS event must be programmed in the port C parallel I/O, not in the SCC itself.

Figure 22-10. SCC UART Interrupt Event Example

SCCE bits are cleared by writing ones; writing zeros has no effect. Unmasked bits must be cleared before the CPM clears an internal interrupt request. Figure 22-11 shows SCCE/SCCM for UART operation.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—			GLR	GLT	—	AB	IDL	GRA	BRKE	BRKS	—	CCR	BSY	TX	RX
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	0xA10 (SCCE1)/0xA14 (SCCM1)															

Figure 22-11. SCC UART Event Register (SCCE) and Mask Register (SCCM)

Table 22-12 describes SCCE fields for UART mode.

Table 22-12. SCCE/SCCM Field Descriptions for UART Mode

Bit	Name	Description
0–2	—	Reserved, should be cleared.
3	GLR	Glitch on Rx. Set when the SCC encounters an Rx clock glitch.
4	GLT	Glitch on transmit. Set when the SCC encounters a Tx clock glitch.
5	—	Reserved, should be cleared.
6	AB	Autobaud. Set when an autobaud lock is detected. The core should rewrite the baud rate generator with the precise divider value. See Section 20.4, “Baud Rate Generators (BRGs).”
7	IDL	Idle sequence status changed. Set when the channel detects a change in the serial line. The line’s real-time status can be read in SCCS[ID]. Idle is entered when a character of all ones is received; it is exited when a zero is received.
8	GRA	Graceful stop complete. Set as soon as the transmitter finishes any buffer in progress after a GRACEFUL STOP TRANSMIT command is issued. It is set immediately if no buffer is in progress.
9	BRKE	Break end. Set when an idle bit is received after a break sequence.
10	BRKS	Break start. Set when the first character of a break sequence is received. Multiple BRKS events are not received if a long break sequence is received.
11	—	Reserved, should be cleared.
12	CCR	Control character received and rejected. Set when a control character is recognized and stored in the receive control character register RCCR.
13	BSY	Busy. Set when a character is received and discarded due to a lack of buffers. In multidrop mode, the receiver automatically enters hunt mode; otherwise, reception continues when a buffer is available. The latest point that an RxBD can be changed to empty and guarantee avoiding the busy condition is the middle of the stop bit of the first character to be stored in that buffer.
14	TX	Tx event. Set when a buffer is sent. If TxBD[CR] = 1, TX is set no sooner than when the last stop bit of the last character in the buffer begins transmission. If TxBD[CR] = 0, TX is set after the last character is written to the Tx FIFO. TX also represents a $\overline{\text{CTS}}$ lost error; check TxBD[CT].
15	RX	Rx event. Set when a buffer is received, which is no sooner than the middle of the first stop bit of the character that caused the buffer to close. Also represents a general receiver error (overrun, $\overline{\text{CD}}$ lost, parity, idle sequence, and framing errors); the RxBD status and control fields indicate the specific error.

22.20 SCC UART Status Register (SCCS)

The SCC UART status register (SCCS), shown in Figure 22-12, monitors the real-time status of RXD. The real-time status of $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ is part of the port C parallel I/O.

Bit	0	1	2	3	4	5	6	7
Field	—							ID
Reset	0000_0000_0000_0000							
R/W	R							
Addr	0xA17 (SCCS1)							

Figure 22-12. SCC Status Register for UART Mode (SCCS)

Table 22-13 describes UART SCCS fields.

Table 22-13. UART SCCS Field Descriptions

Bits	Name	Description
0–6	—	Reserved, should be cleared.
7	ID	Idle status. Set when RXD has been a logic one for at least a full character time. 0 The line is not idle. 1 The line is idle.

22.21 SCC UART Programming Example

The following initialization sequence is for the 9,600 baud, 8 data bits, no parity, and stop bit of the SCC in UART mode assuming a 25-MHz system frequency. BRG1 is used. The controller is configured with $\overline{\text{RTS1}}$, $\overline{\text{CTS1}}$, and $\overline{\text{CD1}}$ active; $\overline{\text{CTS1}}$ acts as an automatic flow-control signal.

1. Configure port A to enable TXD1 and RXD1. Set PAPAR[14,15] and clear PADIR[14,15] and PAODR[14,15].
2. Configure port C to enable $\overline{\text{RTS1}}$, $\overline{\text{CTS1}}$, and $\overline{\text{CD1}}$. Set PCPAR[15] and PCSO[10,11] and clear PCPAR[10,11] and PCDIR[10,11,15].
3. Configure BRG1. 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.
4. Connect BRG1 to SCC1 using the serial interface. Clear SICR[R1CS,T1CS].
5. Initialize the SDMA configuration register (SDCR = 0x0001 for normal operation).
6. Connect the SCC1 to the NMSI. Clear SICR[SC1].
7. Write RBASE and TBASE in the SCC1 parameter RAM to point to the RxBD and TxBD tables in dual-port RAM. Assuming one RxBD at the start of dual-port RAM followed by one TxBD, write RBASE with 0x0000 and TBASE with 0x0008.
8. Write 0x0001 to CPCR to execute the INIT RX AND TX PARAMS command for SCC1. This command updates RBPTR and TBPTR of the serial channel with the new values of RBASE and TBASE.
9. Write RFCR with 0x10 and TFCR with 0x10 for normal operation.
10. Write MRBLR with the maximum number of bytes per Rx buffer. For this case, assume 16 bytes, so MRBLR = 0x0010.
11. Write MAX_IDL with 0x0000 in the parameter RAM to disable the maximum idle functionality for this example.
12. Set BRKCR to 0x0001 so STOP TRANSMIT commands send only one break character.
13. Clear PAREC, FRMEC, NOSEC, and BRKEC in parameter RAM.
14. Clear UADDR1 and UADDR2. They are not used.

15. Clear TOSEQ. It is not used.
16. Write CHARACTER1–8 with 0x8000. They are not used.
17. Write RCCM with 0xC0FF. It is not used.
18. Initialize the RxBD. Assume the Rx buffer is at 0x0000_1000 in main memory. Write 0xB000 to the RxBD[Status and Control], 0x0000 to RxBD[Data Length] (optional), and 0x0000_1000 to RxBD[Buffer Pointer].
19. Initialize the TxBD. Assume the buffer is at 0x0000_2000 in main memory and contains sixteen 8-bit characters. Write 0xB000 to the TxBD[Status and Control], 0x0010 to TxBD[Data Length], and 0x00002000 to TxBD[Buffer Pointer].
20. Write 0xFFFF to SCCE1 to clear any previous events.
21. Write 0x0003 to SCCM1 to allow the TX and RX interrupts.
22. Write 0x4000_0000 to the CPM interrupt mask register (CIMR) to allow SCC1 to generate a system interrupt. The CICR should also be initialized.
23. Write 0x0000_0020 to GSMR_H1 to configure a small Rx FIFO width.
24. Write 0x0002_8004 to GSMR_L1 to configure 16× sampling for transmit and receive, $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ to automatically control transmission and reception (DIAG bits), and the SCC for UART mode. Notice that the transmitter (ENT) and receiver (ENR) have not been enabled yet.
25. Set PSMR1 to 0xB000 to configure automatic flow control using $\overline{\text{CTS}}$, 8-bit characters, no parity, 1 stop bit, and asynchronous SCC UART operation.
26. Write 0x0002_8034 to GSMR_L1 to enable the transmitter and receiver. This ensures that ENT and ENR are enabled last.

Note that after 16 bytes are sent, the transmit buffer is closed. Additionally, the receive buffer is closed after 16 bytes are received. Data received after 16 bytes causes a busy (out-of-buffers) condition because only one RxBD is prepared.

22.22 S-Records Loader Application

This section describes a downloading application that uses an SCC UART controller. The application performs S-record downloads and uploads between a host computer and an intelligent peripheral through a serial asynchronous line. 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. For flow control, each device can transmit XON and XOFF characters, which are not part of the program being uploaded or downloaded.

For simplicity, assume that the line is not multidrop (no addresses are sent) and that each S-record fits into a single buffer. Follow the basic UART initialization sequence above in Section 22.21, “SCC UART Programming Example,” except allow for more and larger buffers and create the control character table as described in Table 22-14.

Table 22-14. UART Control Characters for S-Records Example

Character	Description
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; R should be set. Whenever the core receives a control-character-received (CCR) interrupt and the RCCR contains XOFF, the software should immediately stop transmitting by setting PSMR[FRZ]. This keeps the other station from losing data when it runs out of Rx buffers.
XON	XON should be received after XOFF. E should be cleared and R should be set. PSMR[FRZ] on the transmitter should now be cleared. The CPM 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 S-records, the core must wait for an RX interrupt, indicating that a complete S-record buffer was received. Transmission requires assembling S-records into buffers and linking them to the TxBD table; transmission can be paused when an XOFF character is received. This scheme minimizes the number of interrupts the core receives (one per S-record) and relieves it from continually scanning for control characters.

Chapter 23
SCC HDLC Mode

HDLC (high-level data link control) is one of the most common protocols in the data link layer, layer 2 of the OSI model. Many other common layer 2 protocols, such as SDLC, SS#7, AppleTalk, LAPB, and LAPD, are based on HDLC and its framing structure in particular. Figure 23-1 shows the HDLC framing structure.

HDLC uses a zero insertion/deletion process (bit-stuffing) to ensure that a data bit pattern matching the delimiter flag does not occur in a field between flags. The HDLC frame is synchronous and relies on the physical layer for clocking and synchronization of the transmitter/receiver.

An address field is needed to carry the frame's destination address because the layer 2 frame can be sent over point-to-point links, broadcast networks, packet-switched or circuit-switched systems. An address field is commonly 0, 8, or 16 bits, depending on the data link layer protocol. SDLC and LAPB use an 8-bit address. SS#7 has no address field because it is always used in point-to-point signaling links. LAPD divides its 16-bit address into different fields to specify various access points within one device. LAPD also defines a broadcast address. Some HDLC-type protocols permit 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. The length of the data in the data field depends on the frame protocol. 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 lsb of each octet is sent first; the msb of the CRC is sent first.

HDLC mode is selected for the SCC by writing GSMR_L[MODE] = 0b0000. In a nonmultiplexed modem interface, SCC outputs connect directly to external pins. Modem signals can be supported through port C. The Rx and Tx clocks can be supplied from either the bank of baud rate generators, by the DPLL, or externally. The SCC can also be connected through the TDM channels of the serial interface (SI). In HDLC mode, the SCC becomes an HDLC controller, and consists of separate transmit and receive sections whose operations are asynchronous with the core.

23.1 SCC HDLC Features

The main features of the SCC in HDLC mode are follows:

- Flexible buffers with multiple buffers per frame
- Separate interrupts for frames and buffers (Rx and Tx)
- Received-frames threshold to reduce interrupt overhead
- Can be used with the SCC DPLL
- Four address comparison registers with mask
- Maintenance of five 16-bit error counters
- Flag/abort/idle generation and detection
- Zero insertion/deletion
- 16- or 32-bit CRC-CCITT generation and 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

23.2 SCC HDLC Channel Frame Transmission

The HDLC transmitter is designed to work with little or no core intervention. Once enabled by the core, a transmitter starts sending flags or idles as programmed in the HDLC mode register (PSMR). The HDLC polls the first BD in the TxBD table. When there is a frame to transmit, the SCC fetches the data from memory and starts sending the frame after sending the minimum number of flags specified between frames. When the end of the current buffer is reached and TxBD[L] (last buffer in frame) is set, the CRC and closing flag are appended. In HDLC mode, the lsb of each octet and the msb of the CRC are sent first. Figure 23-1 shows a typical HDLC frame.

Opening Flag	Address	Control	Information (Optional)	CRC	Closing Flag
8 bits	16 bits	8 bits	8*n* bits	16 bits	8 bits

Figure 23-1. HDLC Framing Structure

After a closing flag is sent, the SCC updates the frame status bits of the BD and clears TxBD[R] (buffer ready). At the end of the current buffer, if TxBD[L] is not set (multiple buffers per frame), only TxBD[R] is cleared. Before the SCC proceeds to the next TxBD in the table, an interrupt can be issued if TxBD[I] is set. This interrupt programmability allows the core to intervene after each buffer, after a specific buffer, or after each frame.

The STOP TRANSMIT command can be used to expedite critical data ahead of previously linked buffers or to support efficient error handling. When the SCC receives a STOP TRANSMIT command, it sends idles or flags instead of the current frame until it receives a RESTART TRANSMIT command. The GRACEFUL STOP TRANSMIT command can be used to insert a high-priority frame without aborting the current one—a graceful-stop-complete event is generated in SCCE[GRA] when the current frame is finished. See Section 23.6, “SCC HDLC Commands.”

23.3 SCC HDLC Channel Frame Reception

The HDLC receiver is designed to work with little or no core intervention to perform address recognition, CRC checking, and maximum frame length checking. Received frames can be used to implement any HDLC-based protocol.

Once enabled by the core, the receiver waits for an opening flag character. When it detects the first byte of the frame, the SCC compares the frame address with four user-programmable, 16-bit address registers and an address mask. The SCC compares the received address field with the user-defined values after masking with the address mask. To detect broadcast (all ones) address frames, one address register must be written with all ones.

If an address match is detected, the SCC fetches the next BD and SCC starts transferring the incoming frame to the buffer if it is empty. When the buffer is full, the SCC clears RxBD[E] and generates a maskable interrupt if RxBD[I] is set. If the incoming frame is larger than the current buffer, the SCC continues receiving using the next BD in the table.

During reception, the SCC checks for frames that are too long (using MFLR). When the frame ends, the CRC field is checked against the recalculated value and written to the buffer. RxBD[Data Length] of the last BD in the HDLC frame contains the entire frame length. This also enables software to identify the frames in which the maximum frame length violations occur. The SCC sets RxBD[L] (last buffer in frame), writes the frame status bits, and clears RxBD[E]. It then generates a maskable event (SCCE[RXF]) to indicate a frame was received. The SCC then waits for a new frame. Back-to-back frames can be received with only one shared flag between frames.

The received frames threshold parameter (RFTHR) can be used to postpone interrupts until a specified number of frames is received. This function can be combined with a timer to implement a timeout if fewer than the specified number of threshold frames is received.

Note that the SCC in HDLC mode, or any other synchronous mode, must receive a minimum of eight clocks after the last bit arrives to account for Rx FIFO delay.

23.4 SCC HDLC Parameter RAM

For HDLC mode, the protocol-specific area of the SCC parameter RAM is mapped as in Table 23-1.

Table 23-1. HDLC-Specific SCC Parameter RAM Memory Map

Offset [1]	Name	Width	Description
0x30	—	Word	Reserved
0x34	**C_MASK**	Word	CRC mask. For the 16-bit CRC-CCITT, initialize with 0x0000_F0B8. For 32-bit CRC-CCITT, initialize with 0xDEBB_20E3.
0x38	**C_PRES**	Word	CRC preset. For the 16-bit CRC-CCITT, initialize with 0x0000_FFFF. For 32-bit CRC-CCITT, initialize with 0xFFFF_FFFF.
0x3C	**DISFC**	Hword	Modulo 2^{16} counters maintained by the CPM. Initialize them while the channel is disabled. DISFC (Discarded frame counter) Counts error-free frames discarded due to lack of free buffers. CRCEC (CRC error counter) Includes frames not addressed to the user or frames received in the BSY condition, but does not include overrun errors. ABTSC (Abort sequence counter) NMARC (Nonmatching address received counter) Includes error-free frames only. RETRC (Frame retransmission counter) Counts number of frames resent due to collision.
0x3E	**CRCEC**	Hword	
0x40	**ABTSC**	Hword	
0x42	**NMARC**	Hword	
0x44	**RETRC**	Hword	
0x46	**MFLR**	Hword	Max frame length register. The HDLC compares the incoming HDLC frame's length with the user-defined limit in MFLR. If the limit is exceeded, the rest of the frame is discarded and RxBD[LG] is set in the last BD of that frame. At the end of the frame the SCC reports frame status and frame length in the last RxBD. The MFLR is defined as all in-frame bytes between the opening and closing flags.
0x48	MAX_CNT	Hword	Maximum length counter. A temporary down-counter used to track frame length.
0x4A	**RFTHR**	Hword	Received frames threshold. Used to reduce potential interrupt overhead when each in a series of short HDLC frames causes an SCCE[RXF] event. Setting RFTHR determines the frequency of RXF interrupts, which occur only when the RFTHR limit is reached. Provide enough empty RxBDs for the number of frames specified in RFTHR.
0x4C	RFCNT	Hword	Received frames count. RFCNT is a down-counter used to implement RFTHR.
0x4E	**HMASK**	Hword	Mask register (HMASK) and four address registers (HADDR*n*) for address recognition. The SCC reads the frame address from the HDLC receiver, compares it with the HADDRs, and masks the result with HMASK. Setting an HMASK bit enables the corresponding comparison bit, clearing a bit masks it. When a match occurs, the frame address and data are written to the buffers. When no match occurs and a frame is error-free, the nonmatching address received counter (NMARC) is incremented. The eight low-order bits of HADDR*n* should contain the first address byte after the opening flag. For example, to recognize a frame that begins 0x7E (flag), 0x68, 0xAA, using 16-bit address recognition, HADDR*n* should contain 0xAA68 and HMASK should contain 0xFFFF. For 8-bit addresses, clear the eight high-order HMASK bits. See Figure 23-2..
0x50	**HADDR1**	Hword	
0x52	**HADDR2**	Hword	
0x54	**HADDR3**	Hword	
0x56	**HADDR4**	Hword	
0x58	TMP	Hword	Temporary storage.
0x5A	TMP_MB	Hword	Temporary storage.

1 From SCC base. SCC base = IMMR + 0x3C00 (SCC1)

Figure 23-2 shows 16- and 8-bit address recognition.

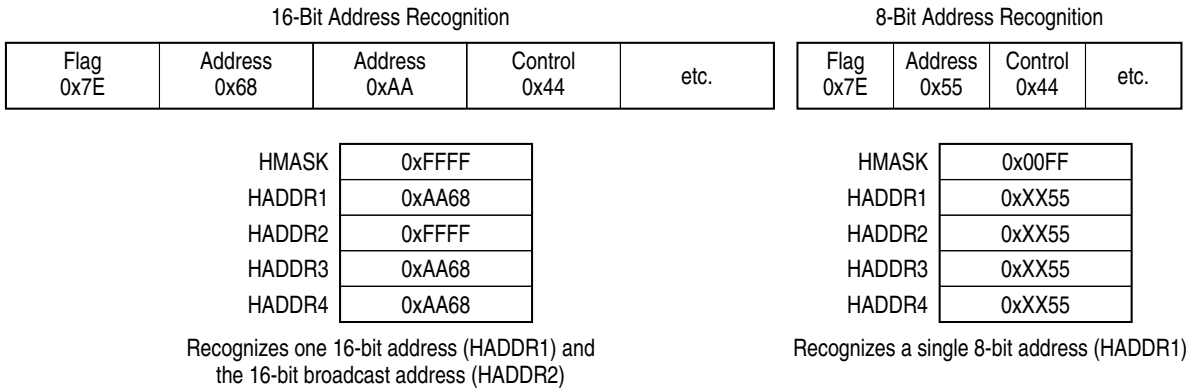


Figure 23-2. HDLC Address Recognition

23.5 Programming the SCC HDLC Controller

HDLC mode is selected for the SCC by writing GSMR_L[MODE] = 0b0000. The HDLC controller uses the same buffer and BD data structure as other modes and supports multibuffer operation and address comparisons. Receive errors are reported through the RxBD; transmit errors are reported through the TxBD.

23.6 SCC HDLC Commands

The transmit and receive commands are issued to the CPM command register (CPCR). Transmit commands are described in Table 23-2.

Table 23-2. Transmit Commands

Command	Description
STOP TRANSMIT	After a hardware or software reset and a channel is enabled in the GSMR, the transmitter starts polling the first BD in the TxBD table every 64 Tx clocks, or immediately if TODR[TOD] = 1, and begins sending data if TxBD[R] is set. If the SCC receives the STOP TRANSMIT command while not transmitting, the transmitter stops polling the BDs. If the SCC receives the command during transmission, transmission is aborted after a maximum of 64 additional bits, the Tx FIFO is flushed, and the current BD pointer TBPTR is not advanced (no new BD is accessed). The transmitter then sends an abort sequence (0x7F) and stops polling the BDs. When not transmitting, the channel sends flags or idles as programmed in the GSMR. Note that if PSMR[MFF] = 1, multiple small frames could be flushed from the Tx FIFO; a GRACEFUL STOP TRANSMIT command prevents this.
GRACEFUL STOP TRANSMIT	Stops transmission smoothly. Unlike a STOP TRANSMIT command, it stops transmission after the current frame is finished or immediately if no frame is being sent. SCCE[GRA] is set when transmission stops. HDLC Tx parameters and Tx BDs can then be updated. TBPTR points to the next TxBD. Transmission begins once TxBD[R] of the next BD is set and a RESTART TRANSMIT command is issued.

Table 23-2. Transmit Commands (continued)

Command	Description
RESTART TRANSMIT	Enables frames to be sent on the transmit channel. The HDLC controller expects this command after a STOP TRANSMIT is issued and the channel in its GSMR is disabled, after a GRACEFUL STOP TRANSMIT command, or after a transmitter error. The transmitter resumes from the current BD.
INIT TX PARAMETERS	Resets the Tx parameters in the parameter RAM. Issue only when the transmitter is disabled. INIT TX AND RX PARAMETERS resets both Tx and Rx parameters.

Receive commands are described in Table 23-3.

Table 23-3. Receive Commands

Command	Description
ENTER HUNT MODE	After a hardware or software reset, once the SCC is enabled in the GSMR, the receiver is automatically enabled and uses the first BD in the RxBD table. While the SCC is looking for the beginning of a frame, that SCC is in hunt mode. The ENTER HUNT MODE command is used to force the HDLC receiver to stop receiving the current frame and enter hunt mode, in which the HDLC continually scans the input data stream for a flag sequence. After receiving the command, the buffer is closed and the CRC is reset. Further frame reception uses the next BD.
CLOSE RXBD	Should not be used in the HDLC protocol.
INIT RX PARAMETERS	Resets the Rx parameters in the parameter RAM.; issue only when the receiver is disabled. Note that INIT TX AND RX PARAMETERS resets both Tx and Rx parameters.

23.7 Handling Errors in the SCC HDLC Controller

The SCC HDLC controller reports frame reception and transmission errors using BDs, error counters, and the SCCE. Transmission errors are described in Table 23-4.

Table 23-4. Transmit Errors

Error	Description
Transmitter Underrun	The channel stops transmitting, closes the buffer, sets TxBD[UN], and generates a TXE interrupt if not masked. Transmission resumes when a RESTART TRANSMIT command is issued. The SCC1 transmit and receive FIFOs are 32 bytes each.
$\overline{\text{CTS}}$ Lost during Frame Transmission	The channel stops transmitting, closes the buffer, sets TxBD[CT], and generates the TXE interrupt if not masked. Transmission resumes after a RESTART TRANSMIT command. If this error occurs on the first or second buffer of the frame and PSMR[RTE] = 1, the channel resends the frame when $\overline{\text{CTS}}$ is reasserted and no error is reported. If collisions are possible, to ensure proper retransmission of multi-buffer frames, the first two buffers of each frame should in total contain more than 36 bytes for SCC1 or 20 bytes for SCC. The channel also increments the retransmission counter RETRC in the parameter RAM.

Reception errors are described in Table 23-5.

Table 23-5. Receive Errors

Error	Description
Overrun	Each SCC maintains an internal FIFO for receiving data. The CPM begins programming the SDMA channel (if the buffer is in external memory) and updating the CRC when a full or partial FIFO's worth of data (according to GSMR_H[RFW]) is received in the Rx FIFO. When an Rx FIFO overrun occurs, the previous byte is overwritten by the next byte. The previous data byte and the frame status are lost. The channel closes the buffer with RxBD[OV] set and generates an RXF interrupt if not masked. The receiver then enters hunt mode. Even if an overrun occurs during a frame whose address is not recognized, an RxBD with data length two is opened to report the overrun and the interrupt is generated.
$\overline{\text{CD}}$ Lost during Frame Reception	Highest priority error. The channel stops frame reception, closes the buffer, sets RxBD[CD], and generates the RXF interrupt if not masked. 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	Occurs when seven or more consecutive ones are received. When this occurs while receiving a frame, the channel closes the buffer, sets RxBD[AB] and generates a maskable RXF interrupt. The channel also increments the abort sequence counter ABTSC. The CRC and nonoctet error status conditions are not checked on aborted frames. The receiver then enters hunt mode.
Nonoctet Aligned Frame	The channel writes the received data to the buffer, closes the buffer, sets RxBD[NO], and generates a maskable RXF interrupt. 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 buffer as follows: msb lsb [ ] [1] [0 0] Valid Data | Nonvalid Data Note that if buffer swapping is used (RFCR[BO] = 0b0x), the figure above refers to the last byte, rather than the last word, of the buffer. The lsb of each octet is sent first while the msb of the CRC is sent first.
CRC	The channel writes the received CRC to the buffer, closes the buffer, sets RxBD[CR], generates a maskable RXF interrupt, and increments the CRC error counter CRCEC. 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.

23.8 HDLC Mode Register (PSMR)

The protocol-specific mode register (PSMR), shown in Figure 23-3, functions as the HDLC mode register.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	NOF				CRC		RTE	—	FSE	DRT	BUS	BRM	MFF	—		
Reset	0															
R/W	R/W															
Address	0xA08 (PSMR1)															

Figure 23-3. HDLC Mode Register (PSMR)

Table 23-6 describes PSMR HDLC fields.

Table 23-6. PSMR HDLC Field Descriptions

Bits	Name	Description
0-3	NOF	Number of flags. Minimum number of flags between or before frames. If NOF = 0b0000, no flags are inserted between frames and the closing flag of one frame is followed by the opening flag of the next frame in the case of back-to-back frames. NOF can be modified on-the-fly.
4–5	CRC	CRC selection. 00 16-bit CCITT-CRC (HDLC). X16 + X12 + X5 + 1. x1 Reserved. 10 32-bit CCITT-CRC (Ethernet and HDLC). X32 + X26 + X23 + X22 + X16 + X12 + X11 + X10 + X8 + X7 + X5 + X4 + X2 + X1 +1.
6	RTE	Retransmit enable. 0 No retransmission. 1 Automatic frame retransmission is enabled. Particularly useful in the HDLC bus protocol and ISDN applications where multiple HDLC controllers can collide. Note that retransmission occurs only if a lost $\overline{\text{CTS}}$ occurs on the first or second buffer of the frame.
7	—	Reserved, should be cleared.
8	FSE	Flag sharing enable. Valid only if GSMR_H[RTSM] = 1. Can be modified on-the-fly. 0 Normal operation. 1 If NOF[0–3] = 0b0000, a single shared flag is sent between back-to-back frames. Other values of NOF[0–3] are decremented by 1. Useful in signaling system #7 applications.
9	DRT	Disable receiver while transmitting. 0 Normal operation. 1 As the SCC sends data, the receiver is disabled and gated by the internal $\overline{\text{RTS}}$. This helps if the HDLC channel is on a multidrop line and the SCC does not need to receive its own transmission.
10	BUS	HDLC bus mode. 0 Normal HDLC operation. 1 HDLC bus operation is selected. See Section 23.14, “HDLC Bus Mode with Collision Detection.”
11	BRM	HDLC bus $\overline{\text{RTS}}$ mode. Valid only 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 Tx 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, which helps when the HDLC bus protocol is being run locally and sent over a long-distance line at the same time. The one-bit delay allows $\overline{\text{RTS}}$ to be used to enable the transmission line buffers so that the electrical effects of collisions are not sent over the transmission line.
12	MFF	Multiple frames in Tx FIFO. The receiver is not affected. 0 Normal operation. The Tx FIFO must never contain more than one HDLC frame. The $\overline{\text{CTS}}$ lost status is reported accurately on a per-frame basis. 1 The Tx FIFO can hold multiple frames, but lost $\overline{\text{CTS}}$ may not be reported on the buffer/frame it occurred on. This can improve performance of HDLC transmissions of small back-to-back frames or when the number of flags between frames should be limited.
13–15	—	Reserved, should be cleared.

23.9 SCC HDLC Receive Buffer Descriptor (RxBD)

The CPM uses the RxBD, shown in Figure 23-4, to report on data received for each buffer.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**E**	—	**W**	**I**	L	F	**CM**	—	DE	—	LG	NO	AB	CR	OV	CD
Offset + 2	Data Length															
Offset + 4	**Rx Buffer Pointer**															
Offset + 6																

Figure 23-4. SCC HDLC Receive Buffer Descriptor (RxBD)

Table 23-7 describes HDLC RxBD status and control fields.

Table 23-7. SCC HDLC RxBD Status and Control Field Descriptions

Bits	Name	Description
0	**E**	Empty. 0 The buffer is full or reception stopped because of an error. The core can read or write to any fields of this RxBD. The CPM does not use this BD while E = 0. 1 The buffer is not full. The CP controls the BD and buffer. The core should not update the BD.
1	—	Reserved, should be cleared.
2	**W**	Wrap (last BD in the RxBD table). 0 Not the last BD in the table. 1 Last BD in the table. After this buffer is used, the CPM receives incoming data using the BD pointed to by RBASE. The number of BDs in this table are programmable and determined only by RxBD[W] and overall space constraints of the dual-port RAM.
3	**I**	Interrupt. 0 SCCE[RXB] is not set after this buffer is used; SCCE[RXF] is unaffected. 1 SCCE[RXB] or SCCE[RXF] is set when the SCC uses this buffer.
4	L	Last buffer in frame. 0 Not the last buffer in frame. 1 Last buffer in frame. Indicates reception of a closing flag or an error, in which case one or more of the CD, OV, AB, and LG bits are set. The SCC writes the number of frame octets to the data length field.
5	F	First in frame. 0 Not the first buffer in a frame. 1 First buffer in a frame.
6	**CM**	Continuous mode. Note that RxBD[E] is cleared if an error occurs during reception, regardless of CM. 0 Normal operation. 1 RxBD[E] is not cleared by the CPM after this BD is closed, allowing the associated buffer to be overwritten next time the CPM accesses it.
7	—	Reserved, should be cleared.
8	DE	DPLL error. Set when a DPLL error occurs while this buffer is being received. DE is also set due to a missing transition when using decoding modes in which a transition is required for every bit. Note that when a DPLL error occurs, the frame closes and error checking halts.
9	—	Reserved, should be cleared.

Table 23-7. SCC HDLC RxBD Status and Control Field Descriptions (continued)

Bits	Name	Description
10	LG	Rx frame length violation. Set when a frame larger than the maximum defined for this channel is recognized. Only the maximum-allowed number of bytes (MFLR) is written to the buffer. This event is not reported until the buffer is closed, SCCE[RXF] is set, and the closing flag is received. The total number of bytes received between flags is still written to the data length field.
11	NO	Rx nonoctet aligned frame. Set when a received frame contains a number of bits not divisible by eight.
12	AB	Rx abort sequence. Set when at least seven consecutive ones are received during frame reception.
13	CR	Rx CRC error. Set when a frame contains a CRC error. CRC bytes received are always written to the Rx buffer.
14	OV	Overrun. Set when a receiver overrun occurs during frame reception.
15	CD	Carrier detect lost (NMSI mode only). Set when $\overline{CD}$ is negated during frame reception.

Data length and buffer pointer fields are described in Section 21.3, “SCC Buffer Descriptors (BDs).” Because HDLC is a frame-based protocol, RxBD[Data Length] of the last buffer of a frame contains the total number of frame bytes, including the 2 or 4 bytes for CRC. Figure 23-5 shows an example of how RxBDs are used in receiving.

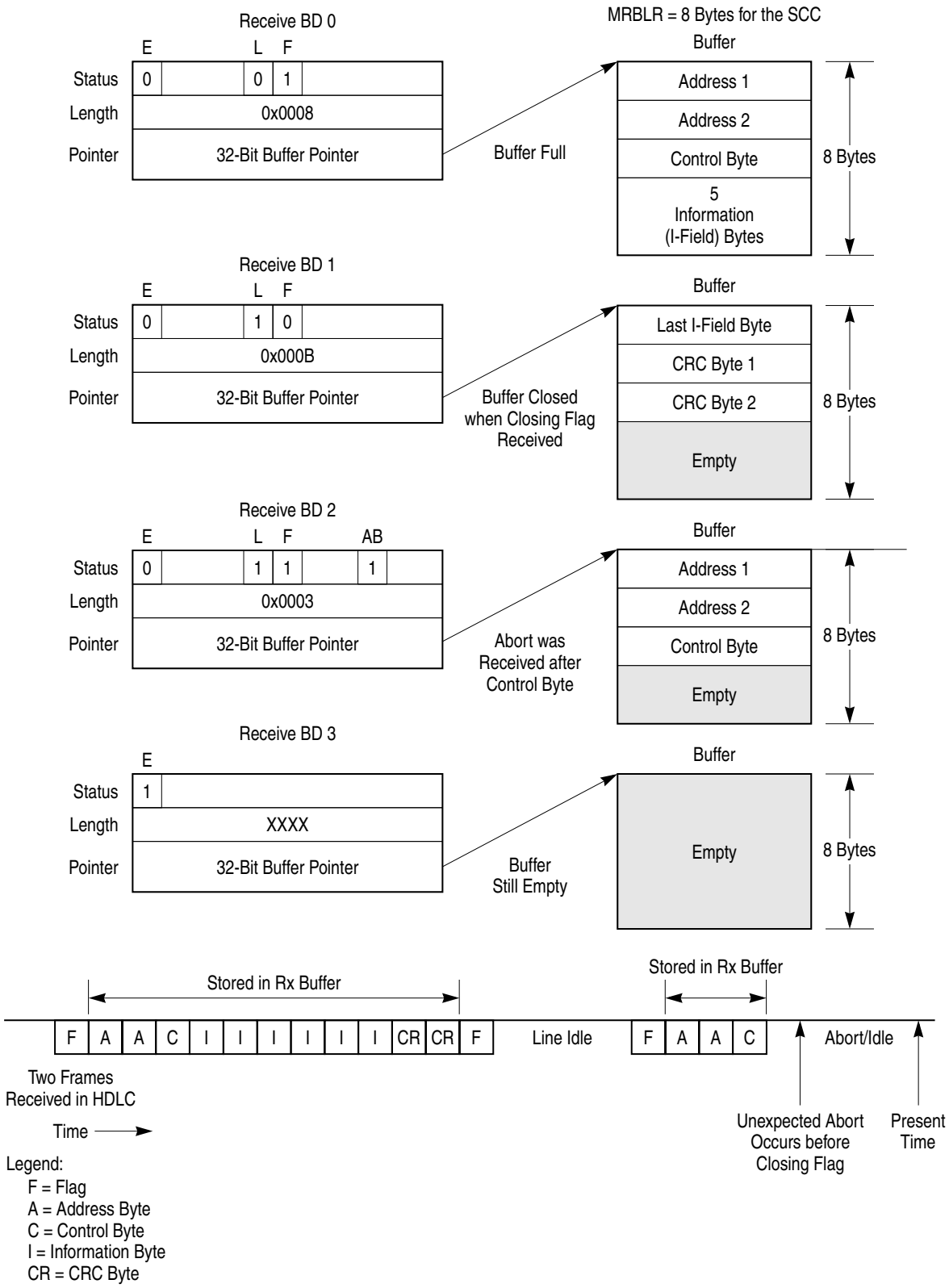


Figure 23-5. SCC HDLC Receiving using RxBDs

23.10 SCC HDLC Transmit Buffer Descriptor (TxBD)

The CPM uses the TxBD, shown in Figure 23-6, to confirm transmissions and indicate error conditions.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**R**	—	**W**	**I**	**L**	**TC**	**CM**	—							UN	CT
Offset + 2	**Data Length**															
Offset + 4	**Tx Buffer Pointer**															
Offset + 6																

Figure 23-6. SCC HDLC Transmit Buffer Descriptor (TxBD)

Table 23-8 describes HDLC TxBD status and control fields.

Table 23-8. SCC HDLC TxBD Status and Control Field Descriptions

Bits	Name	Description
0	**R**	Ready. 0 The buffer is not ready for transmission. Both the buffer and the BD can be updated. The CPM clears R after the buffer is sent or an error is encountered. 1 The buffer has not been sent or is being sent and the BD cannot be updated.
1	—	Reserved, should be cleared.
2	**W**	Wrap (last BD in TxBD table). 0 Not the last BD in the table. 1 Last BD in the BD table. After this buffer is used, the CPM sends data using the BD pointed to by TBASE. The number of TxBDs in this table is determined by TxBD[W] and the space constraints of the dual-port RAM.
3	**I**	Interrupt. 0 No interrupt is generated after this buffer is processed. 1 SCCE[TXB] or SCCE[TXE] is set when this buffer is processed, causing interrupts if not masked.
4	**L**	Last. 0 Not the last buffer in the frame. 1 Last buffer in the frame.
5	**TC**	Tx CRC. Valid only when TxBD[L] = 1. 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.
6	**CM**	Continuous mode. 0 Normal operation. 1 The CP does not clear TxBD[R] after this BD is closed allowing the buffer to be resent the next time the CP accesses this BD. However, TxBD[R] is cleared if an error occurs during transmission, regardless of CM.
7–13	—	Reserved, should be cleared.
14	UN	Underrun. Set after the SCC sends a buffer and a transmitter underrun occurred.
15	CT	CTS lost. Indicates when $\overline{\text{CTS}}$ in NMSI mode or layer 1 grant is lost during frame transmission. If data from more than one buffer is currently in the FIFO when this error occurs, the HDLC writes CT in the current BD after sending the buffer.

The data length and buffer pointer fields are described in Section 21.3, “SCC Buffer Descriptors (BDs).”

23.11 HDLC Event Register (SCCE)/HDLC Mask Register (SCCM)

The SCC event register (SCCE) is used as the HDLC event register to report events recognized by the HDLC channel and to generate interrupts. When an event is recognized, the SCC sets the corresponding SCCE bit. Interrupts generated through SCCE can be masked in the SCC mask register (SCCM) which has the same bit format as the SCCE. Setting an SCCM bit enables the corresponding interrupt; clearing a bit masks it. SCCE bits are cleared by writing ones; writing zeros has no effect. All unmasked bits must be cleared before the CPM clears the internal interrupt request. Figure 23-7 shows SCCE/SCCM for HDLC operation.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—			GLR	GLT	DCC	FLG	IDL	GRA	—		TXE	RXF	BSY	TXB	RXB
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	0xA10 (SCCE1)/0xA14 (SCCM1)															

Figure 23-7. HDLC Event Register (SCCE)/HDLC Mask Register (SCCM)

Table 23-9 describes SCCE/SCCM fields.

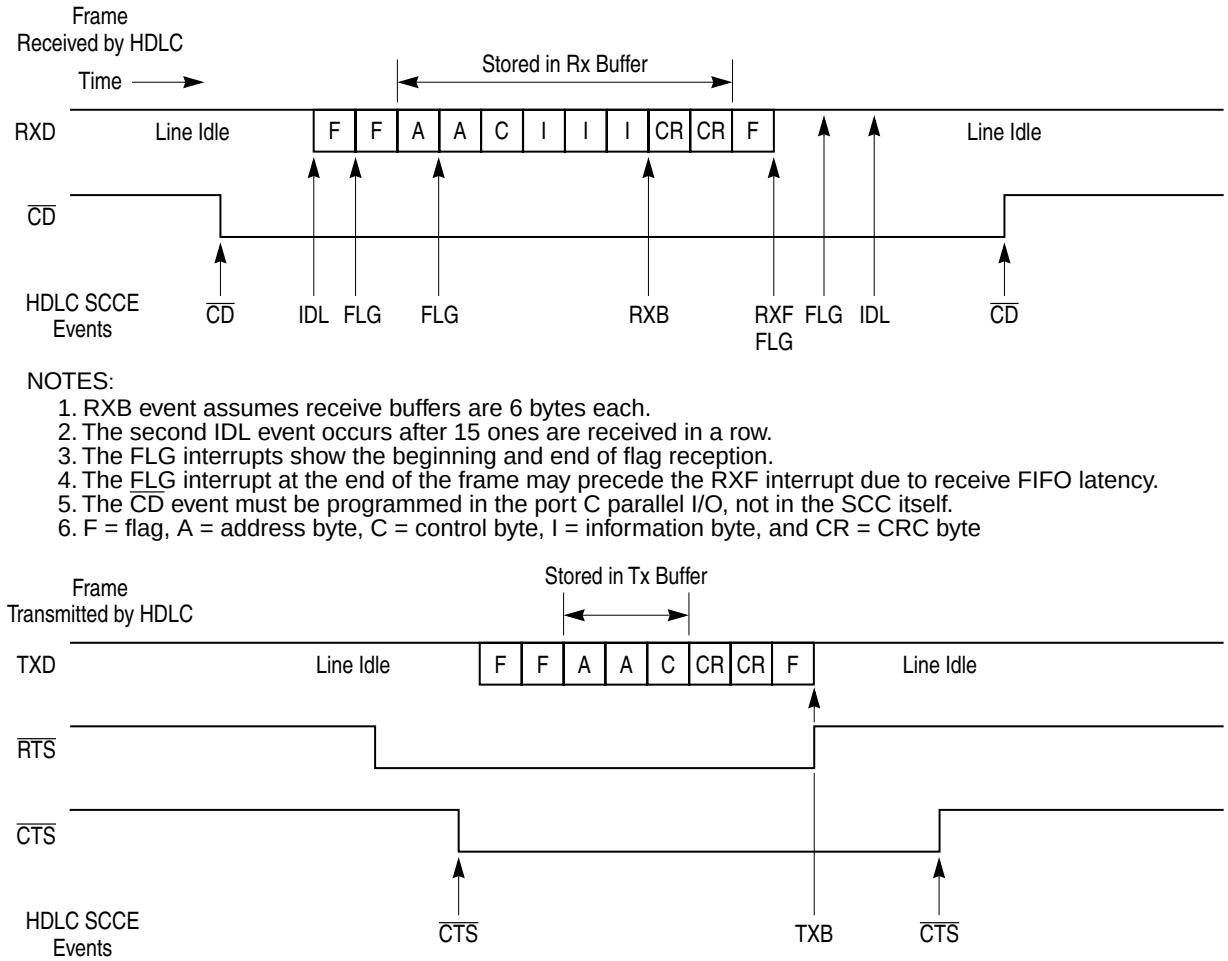
Table 23-9. SCCE/SCCM Field Descriptions

Bits	Name	Description
0–2	—	Reserved, should be cleared.
3, 4	GLR/ GLT	Glitch on Rx/Tx. Set when the SCC detects a clock glitch on the receive/transmit clock. See Section 21.4.6, “Clock Glitch Detection.”
5	DCC	DPLL carrier sense changed. Set when the carrier sense status generated by the DPLL changes. Real-time status can be read in SCCS[CS]. This is not the $\overline{CD}$ status reported in port C. Valid only when the DPLL is used.
6	FLG	Flag status. Set when the SCC stops or starts receiving HDLC flags. Real-time status can be read in SCCS[FG].
7	IDL	Idle sequence status changed. Set when HDLC line status changes. Real-time status of the line can be read in SCCS[ID].
8	GRA	Graceful stop complete. A GRACEFUL STOP TRANSMIT command completed execution. Set as soon as the transmitter has sent a frame in progress when the command was issued. Set immediately if no frame was in progress when the command was issued.
9–10	—	Reserved, should be cleared.
11	TXE	Tx error. Indicates an error ($\overline{CTS}$ lost or underrun) has occurred on the transmitter channel.

Table 23-9. SCCE/SCCM Field Descriptions (continued)

Bits	Name	Description
12	RXF	Rx frame. Set when the number of receive frames specified in RFTHR are received on the HDLC channel. It is set no sooner than two clocks after the last bit of the closing flag is received. This event is not maskable via the RxBD[I] bit.
13	BSY	Busy condition. Indicates a frame arrived but was discarded due to a lack of buffers.
14	TXB	Transmit buffer. Enabled by setting TxBD[I]. TXB is set when a buffer is sent on the HDLC channel. For the last buffer in the frame, TXB is not set before the last bit of the closing flag begins its transmission; otherwise, it is set after the last byte of the buffer is written to the Tx FIFO.
15	RXB	Receive buffer. Enabled by setting RxBD[I]. RXB is set when the HDLC channel receives a buffer that is not the last in a frame.

Figure 23-8 shows interrupts that can be generated using the HDLC protocol.



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 the SCC 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 the SCC itself.

Figure 23-8. SCC HDLC Interrupt Event Example

23.12 SCC HDLC Status Register (SCCS)

The SCC status register (SCCS), shown in Figure 23-9, permits monitoring of real-time status conditions on RXD. The real-time status of $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ are part of the port C parallel I/O.

Bit	0	1	2	3	4	5	6	7
Field	—					FG	CS	ID
Reset	0000_0000							
R/W	R							
Addr	0xA17 (SCCS1)							

Figure 23-9. SCC HDLC Status Register (SCCS)

Table 23-10 describes HDLC SCCS fields.

Table 23-10. HDLC SCCS Field Descriptions

Bits	Name	Description
0–4	—	Reserved, should be cleared.
5	FG	Flags. The line is checked after the data has been decoded by the DPLL. 0 HDLC flags are not being received. The most recently received 8 bits are examined every bit time to see if a flag is present. 1 HDLC flags are being received. FG is set as soon as an HDLC flag (0x7E) is received on the line. Once it is set, it remains set at least 8 bit times and the next eight received bits are examined. If another flag occurs, FG stays set for at least another eight bits. If not, it is cleared and the search begins again.
6	CS	Carrier sense (DPLL). 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.
7	ID	Idle status. 0 The line is busy. 1 Set when RXD is a logic 1 (idle) for 15 or more consecutive bit times. It is cleared after a single logic 0 is received.

23.13 SCC HDLC Programming Examples

The following sections show examples for programming the SCC in HDLC mode. The first example uses an external clock. The second example implements Manchester encoding.

23.13.1SCC HDLC Programming Example #1

The following initialization sequence is for the SCC HDLC channel with an external clock. $\overline{\text{RTS1}}$, $\overline{\text{CTS1}}$, and $\overline{\text{CD1}}$ are active; CLK3 is used for both the HDLC receiver and transmitter.

1. Configure port A to enable TXD1 and RXD1. Set PAPAR[14,15] and clear PADIR[14,15] and PAODR[14,15].
2. Configure port C to enable $\overline{\text{RTS1}}$, $\overline{\text{CTS1}}$, and $\overline{\text{CD1}}$. Set PCPAR[15] and PCSO[10,11] and clear PCPAR[10,11] and PCDIR[10,11,15].
3. Configure port A to enable CLK3. Set PAPAR[5] and clear PADIR[5].
4. Connect CLK3 to SCC1 using the SI. Write 0b110 to SICR[R1CS] and SICR[T1CS].
5. Connect the SCC1 to the NMSI (its own set of pins) and clear SICR[SC1].
6. Write 0x0001 to the SDCR to initialize the SDMA configuration register.
7. Write RBASE and TBASE in the SCC1 parameter RAM to point to the RxBD and TxBD tables in dual-port RAM. Assuming one RxBD at the start of dual-port RAM and one TxBD following it, write RBASE with 0x0000 and TBASE with 0x0008.
8. Write 0x0001 to CPCR to execute the INIT RX AND TX PARAMS command for SCC1. This command updates RBPTR and TBPTR of the serial channel with the new values of RBASE and TBASE.
9. Write RFCR with 0x10 and TFCR with 0x10 for normal operation.
10. Write MRBLR with the maximum number of bytes per Rx buffer. Choose 256 bytes (MRBLR = 0x0100) so an entire Rx frame can fit in one buffer.
11. Write C_MASK with 0x0000F0B8 to comply with 16-bit CCITT-CRC.
12. Write C_PRES with 0x0000FFFF to comply with 16-bit CCITT-CRC.
13. Clear DISFC, CRCEC, ABTSC, NMARC, and RETRC for clarity.
14. Write MFLR with 0x0100 so the maximum frame size is 256 bytes.
15. Write RFTHR with 0x0001 to allow interrupts after each frame.
16. Write HMASK with 0x0000 to allow all addresses to be recognized.
17. Clear HADDR1–HADDR4 for clarity.
18. Initialize the RxBD. Assume the buffer is at 0x0000_1000 in main memory. RxBD[Status and Control]= 0xB000, RxBD[Data Length] = 0x0000 (not required), and RxBD[Buffer Pointer] = 0x0000_1000.
19. Initialize the TxBD. Assume the Tx data frame is at 0x0000_2000 in main memory and contains five 8-bit characters. TxBD[Status and Control] = 0xBC00, TxBD[Data Length] = 0x0005, and TxBD[Buffer Pointer] = 0x0000_2000.
20. Write 0xFFFF to SCCE to clear any previous events.

21. Write 0x001A to SCCM to allow TXE, RXF, and TXB interrupts.
22. Write 0x4000_0000 to the CPM interrupt mask register (CIMR) to allow SCC1 to generate a system interrupt. The CICR should also be initialized.
23. Write 0x0000_0000 to GSMR_H1 to enable normal $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ behavior with idles (not flags) between frames.
24. Write 0x0000_0000 to GSMR_L1 to configure $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ to control transmission and reception in HDLC mode. Normal Tx clock operation is used. Notice that the transmitter (ENT) and receiver (ENR) have not been enabled. If inverted HDLC operation is preferred, set RINV and TINV.
25. Write 0x0000 to PSMR1 to configure one opening and one closing flag, 16-bit CCITT-CRC, and prevent multiple frames in the FIFO.
26. Write 0x00000030 to GSMR_L1 to enable the transmitter and receiver. This additional write ensures that ENT and ENR are enabled last.

Note that after 5 bytes and CRC have been sent, the Tx buffer is closed; the Rx buffer is closed after a frame is received. Frames larger than 256 bytes cause a busy (out-of-buffers) condition because only one RxBD is prepared.

23.13.2 SCC HDLC Programming Example #2

The following sequence initializes an HDLC channel that uses the DPLL in a Manchester encoding. Provide a clock which is 16× the chosen bit rate of CLK3. Then connect CLK3 to the HDLC transmitter and receiver. (A baud rate generator could be used instead.) Configure SCC1 to use $\overline{\text{RTS1}}$, $\overline{\text{CTS1}}$, and $\overline{\text{CD1}}$.

1. Follow steps 1–23 in example #1 above.
2. Write 0x004A_A400 to GSMR_L1 to make carrier sense always active, a 16-bit preamble of ‘01’ patterns, 16× operation of the DPLL and Manchester encoding for the receiver and transmitter, and HDLC mode. $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ should be configured to control transmission and reception. Do not set GSMR[ENT, ENR].
3. Write 0x0000 to PSMR1 to use one opening and one closing flag and 16-bit CCITT-CRC and to reject multiple frames in the FIFO.
4. Write 0x004A_A430 to GSMR_L1 to enable the transmitter and receiver. This additional write to GSMR_L1 ensures that ENT and ENR are enabled last.

23.14 HDLC Bus Mode with Collision Detection

The HDLC controller includes an option for hardware collision detection and retransmission on an open-drain connected HDLC bus, referred to as HDLC bus mode. Most HDLC-based controllers provide only point-to-point communications; however, the HDLC bus enhancement allows implementation of an HDLC-based LAN and other point-to-multipoint configurations. The HDLC bus is based on 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 does not fully comply with I.430 or T1.605 and cannot replace devices that implement these protocols. Instead, it is more suited to non-ISDN LAN and point-to-multipoint configurations.

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 way to connect eight terminals over the D-channel of the S/T ISDN bus. The layer 2 protocol is a variant of HDLC, called LAPD. However, at layer 1, a method is provided to allow the eight terminals to send frames to the switch through the physical S/T bus.

To determine whether a channel is clear, the S/T interface device looks at an echo bit on the line designed to echo the last bit sent on the D channel. Depending on the class of terminal and the context, an S/T interface device waits for 7–10 ones on the echo bit before letting the LAPD frame begin transmission, after which the S/T interface monitors transmitted data. As long as the echo bit matches the sent data, transmission continues. If the echo bit is ever 0 when the transmit bit is 1, a collision occurs between terminals; the station(s) that sent a zero stops transmitting. The station that sent a 1 continues as normal.

The I.430 and T1.605 standards provide a physical layer protocol that allows multiple terminals to share one physical connection. These protocols handle collisions efficiently because one station can always complete its transmission, at which point, it lowers its own priority to give other devices fair access to the physical connection.

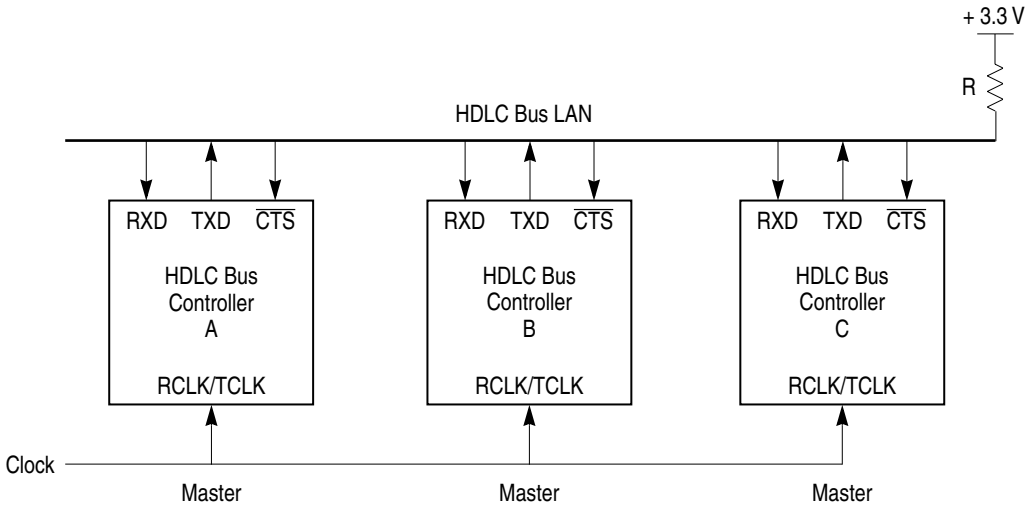
The HDLC bus differs from the I.430 and T1.605 standards as follows:

- The HDLC bus uses a separate input signal rather than the echo bit to monitor data; the transmitted data is simply connected to the $\overline{\text{CTS}}$ input.
- The HDLC bus is a synchronous, digital open-drain connection for short-distance configurations, rather than the more complex S/T interface.
- Any HDLC-based frame protocol can be used at layer 2, not just LAPD.
- HDLC bus devices wait 8–10 rather than 7–10 bit times before transmitting. (HDLC bus has only one class.)

The collision-detection mechanism supports only:

- NRZ-encoded data
- A common synchronous clock for all receivers and transmitters
- Non-inverted data (GSMR[RINV, TINV] = 0)
- Open-drain connection with no external transceivers

Figure 23-10 shows the most common HDLC bus LAN configuration, a multimaster configuration. A station can transfer data to or from any other LAN station. Transmissions are half-duplex, which is typical in LANs.

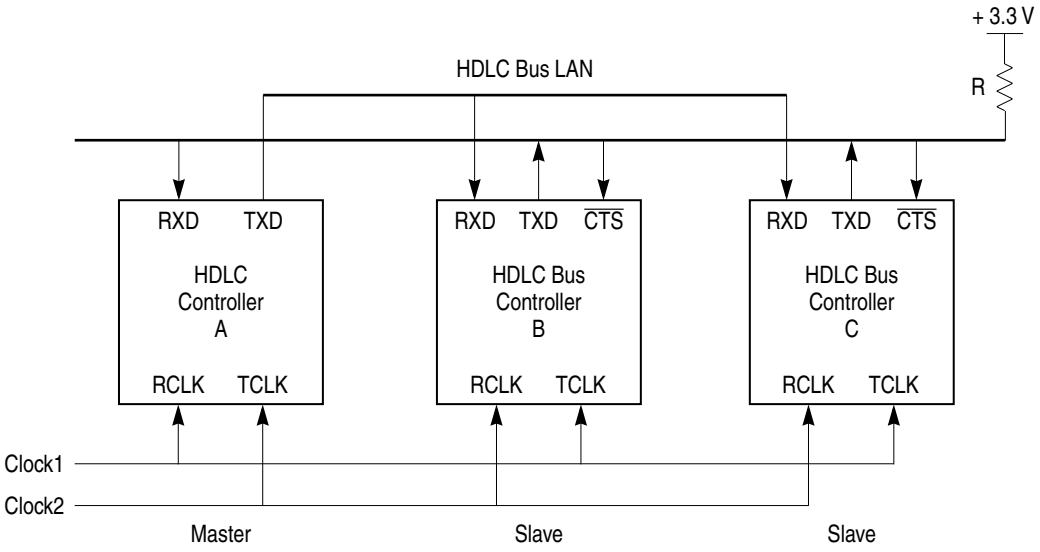


NOTES:
1. Transceivers may be used to extend the LAN size.
2. The TXD pins of slave devices should be configured to open-drain in the port C parallel I/O port.
3. Clock is a common RCLK/TCLK for all stations.

Figure 23-10. Typical HDLC Bus Multimaster Configuration

In single-master configuration, a master station transmits to any slave station without collisions. Slaves communicate only 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 resent 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 23-11 shows the single-master configuration.



NOTES:
1. Transceivers may be used to extend the LAN size.
2. The TXD pins of slave devices should be configured to open-drain in the port C parallel I/O port.
3. Clock1 is the master RCLK and the slave TCLK.
4. Clock2 is the master TCLK and the slave RCLK.

Figure 23-11. Typical HDLC Bus Single-Master Configuration

23.14.1 HDLC Bus Features

The main features of the HDLC bus are as follows:

- Superset of the HDLC controller features
- Automatic HDLC bus access
- Automatic retransmission in case of collision
- May be used with the NMSI or a TDM bus
- Delayed $\overline{\text{RTS}}$ mode

23.14.2 Accessing the HDLC Bus

The HDLC bus protocol ensures orderly bus control when multiple transmitters attempt simultaneous access. The transmitter sending a zero bit at the time of collision completes the transmission. If a station sends out an opening flag (0x7E) while another station is already sending, the collision is always detected within the first byte, because the transmission in progress is using zero bit insertion to prevent flag imitation.

While in the active condition (ready to transmit), the HDLC bus controller monitors the bus using $\overline{\text{CTS}}$. It counts the one bits on $\overline{\text{CTS}}$. When eight consecutive ones are counted, the HDLC bus controller starts transmitting on the line; if a zero is detected, the internal counter is cleared. During transmission, data is continuously compared with the external

bus using $\overline{CTS}$. $\overline{CTS}$ is sampled halfway through the bit time using the rising edge of the Tx clock. If the transmitted bit matches the received $\overline{CTS}$ bus sample, transmission continues. However, if the received $\overline{CTS}$ sample is 0 and the transmitted bit is 1, transmission stops after that bit and waits for an idle line before attempting retransmission. Since the HDLC bus uses a wired-OR scheme, a transmitted zero has priority over a transmitted 1. Figure 23-12 shows how $\overline{CTS}$ is used to detect collisions.

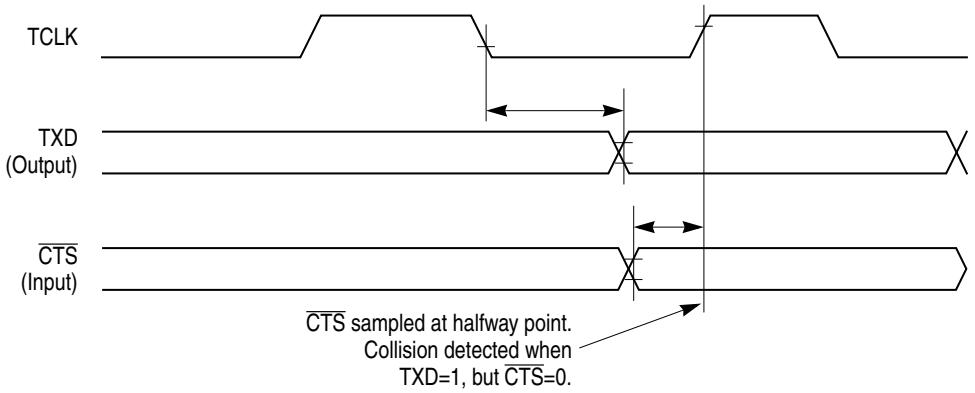


Figure 23-12. Detecting an HDLC Bus Collision

If both the destination address and source address are included in the HDLC frame, then a predefined priority of stations results; if two stations begin to transmit simultaneously, they necessarily detect a collision no later than the end of the source address.

The HDLC bus priority mechanism ensures that stations share the bus equally. To minimize idle time between messages, a station normally waits for eight one bits on the line before attempting transmission. After successfully sending a frame, a station waits for 10 rather than eight consecutive one bits before attempting another transmission. This mechanism ensures that another station waiting to transmit acquires the bus before a station can transmit twice. When a low priority station detects 10 consecutive ones, it tries to transmit; if it fails, it reinstates the high priority of waiting for only eight ones.

23.14.3Increasing Performance

Because it uses a wired-OR configuration, HDLC bus performance is limited by the rise time of the one bit. To increase performance, give the one bit more rise time by using a clock that is low longer than it is high, as shown in Figure 23-13.

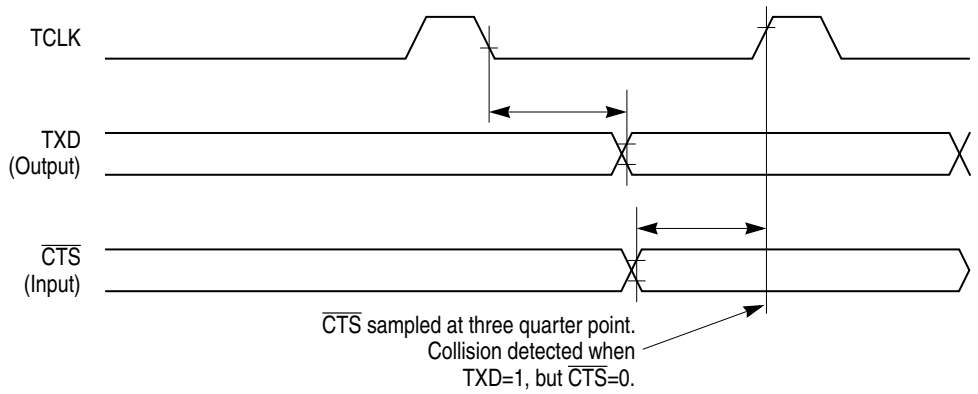
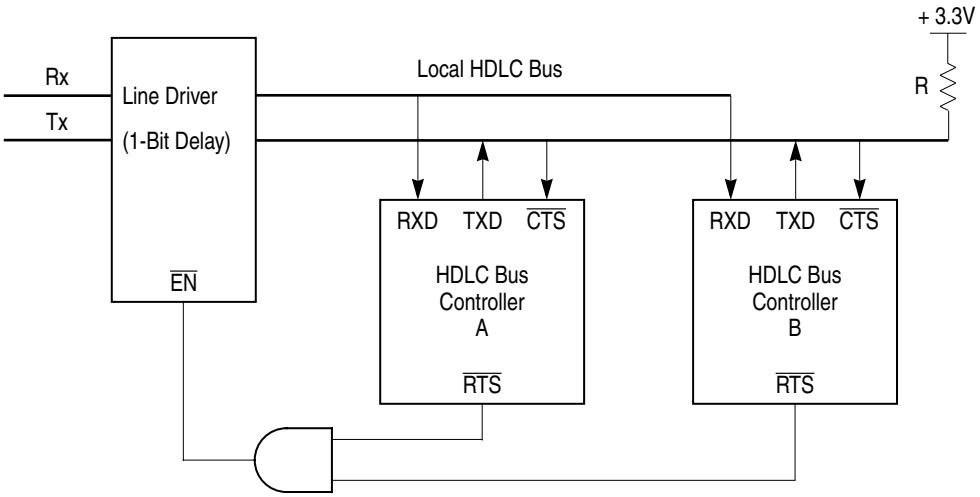


Figure 23-13. Nonsymmetrical Tx Clock Duty Cycle for Increased Performance

23.14.4 Delayed RTS Mode

Figure 23-14 shows local HDLC bus controllers using a standard transmission line and a local bus. The controllers do not communicate with each other but with a station on the transmission line; yet the HDLC bus protocol controls access to the transmission line.



NOTES:
1. The TXD pins of slave devices should be configured to open-drain in the port C parallel I/O port.
2. The RTS pins of each HDLC bus controller are configured to delayed RTS mode.

Figure 23-14. HDLC Bus Transmission Line Configuration

Normally, RTS goes active at the beginning of the opening flag's first bit. Setting PSMR[BRM] delays RTS by one bit, which is useful when the HDLC bus connects multiple local stations to a transmission line. If the transmission line driver has a one-bit delay, the delayed RTS can be used to enable the output of the line driver. As a result, the electrical effects of collisions are isolated locally. Figure 23-15 shows RTS timing.

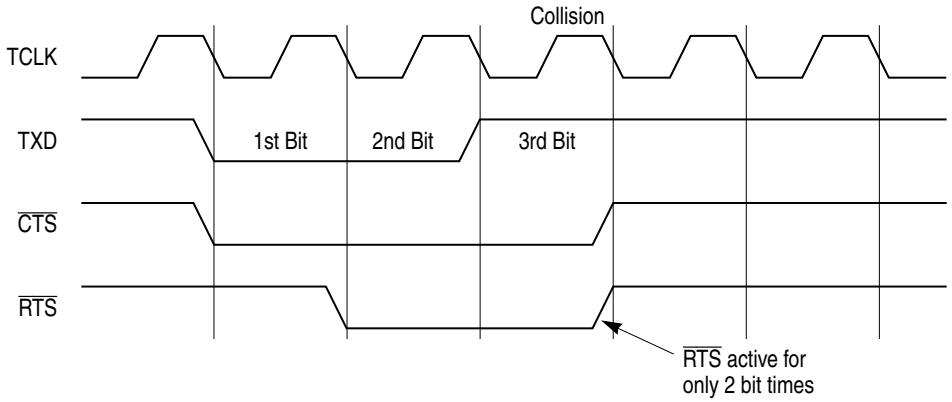
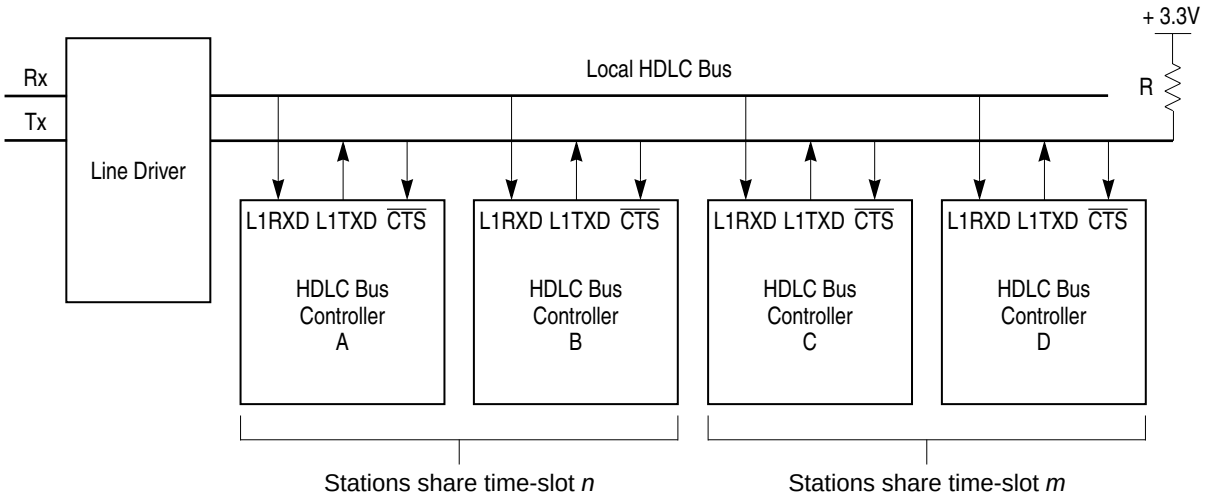


Figure 23-15. Delayed RTS Mode

23.14.5 Using the Time-Slot Assigner (TSA)

HDLC bus controllers can be used with a time-division multiplexed transmission line and a local bus, as shown in Figure 23-16. Local stations use time slots to communicate over the TDM transmission line; stations that share a time slot use the HDLC bus protocol to control access to the local bus.



NOTES:
1. All TXD pins of slave devices should be configured to open-drain in the port C parallel I/O port.
2. The TSA in the SI of each station is used to configure the preferred time slot.
3. The choice of the number of stations to share a time slot is user-defined. It is two in this example.

Figure 23-16. HDLC Bus TDM Transmission Line Configuration

23.14.6 HDLC Bus Protocol Programming

The HDLC bus is implemented using the SCC in HDLC mode with bus-specific options selected in the PSMR and GSMR, as outlined below. See also Section 23.5, “Programming the SCC HDLC Controller.”

23.14.6.1 Programming GSMR and PSMR for the HDLC Bus Protocol

To program the protocol-specific mode register (PSMR), set the bits as described below:

- Configure NOF as preferred
- Set RTE and BUS to 1
- Set BRM to 1 if delayed $\overline{\text{RTS}}$ is desired
- Configure CRC to 16-bit CRC CCITT (0b00).
- Configure other bits to zero or default.

To program the general SCC mode register (GSMR), set the bits as described below:

- Set MODE to HDLC mode (0b0000).
- Configure CTSS to 1 and all other bits to zero or default.
- Configure the DIAG bits for normal operation (0b00).
- Configure RDCR and TDCR for 1× clock (0b00).
- Configure TENC and RENC for NRZ (0b000).
- Clear RTSM to send idles between frames.
- Set GSMR_L[ENT, ENR] as the last step to begin operation.

23.14.6.2 HDLC Bus Controller Programming Example

Except for the above discussion in Section 23.14.6.1, “Programming GSMR and PSMR for the HDLC Bus Protocol,” use the example in Section 23.13.1, “SCC HDLC Programming Example #1.”

Chapter 24
SCC 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, AppleTalk protocols have been most closely associated with the LocalTalk physical and link-layer protocol, an HDLC-based protocol that runs at 230.4 kbps. In this manual, the term ‘AppleTalk controller’ assumes the support that the MPC855T provides for LocalTalk protocol. The AppleTalk controller provides 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.

24.1 Operating the LocalTalk Bus

A LocalTalk frame, shown in Figure 24-1, is basically a modified HDLC frame.

Sync Sequence	HDLC Flags	Destination Address	Source Address	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 24-1. LocalTalk Frame Format

First, a synchronization sequence of more than three bits is sent. This sequence consists of at least one logical one bit (FM0 encoded) followed by two bit times or more 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, allowing bit, byte, and frame delineation or detection. Two bytes of address, destination, and source are sent next, followed by a byte of control and 0–600 data bytes. Next, two bytes of CRC (the common 16-bit CRC-CCITT polynomial referenced in the HDLC standard protocol) 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 from 0x01–0x7F are data frames; control byte values from 0x80–0xFF are control frames. Four control frames are defined:

- ENQ—Enquiry
- ACK—Enquiry acknowledgment
- RTS—Request to send a data frame
- CTS—Clear to send a data frame

Frames are sent in groups known as dialogs, which are handled by the software. For instance, to transfer a data frame, three frames are sent over the network. An RTS frame (not to be confused with the RS-232 $\overline{\text{RTS}}$ pin) is sent to request the network, 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. After a dialog begins, other nodes cannot start sending until the dialog is complete. Frames within a dialog are sent with a maximum interframe gap (IFG) of 200 µs. Although the LocalTalk specification does not state it, there is also a minimum recommended IFG of 50 µs. Dialogs must be separated by a minimum interdialog gap (IDG) of 400 µs. In general, these gaps are implemented by the software.

Depending on the protocol, collisions should be encountered only during RTS and ENQ frames. Once frame transmission begins, it is fully sent, regardless of whether it collides with another frame. ENQ frames are infrequent and are sent only 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 logical 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.

24.2 Features

The following list summarizes the features of the SCC 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 SCCE[DCC] interrupts
- Reception is automatically disabled while sending a frame
- Transmit-on-demand feature expedites frames

- Connects directly to an RS-422 transceiver

24.3 Connecting to AppleTalk

As shown in Figure 24-2, the MPC855T connects to LocalTalk, and, using TXD, $\overline{\text{RTS}}$, and RXD, 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.

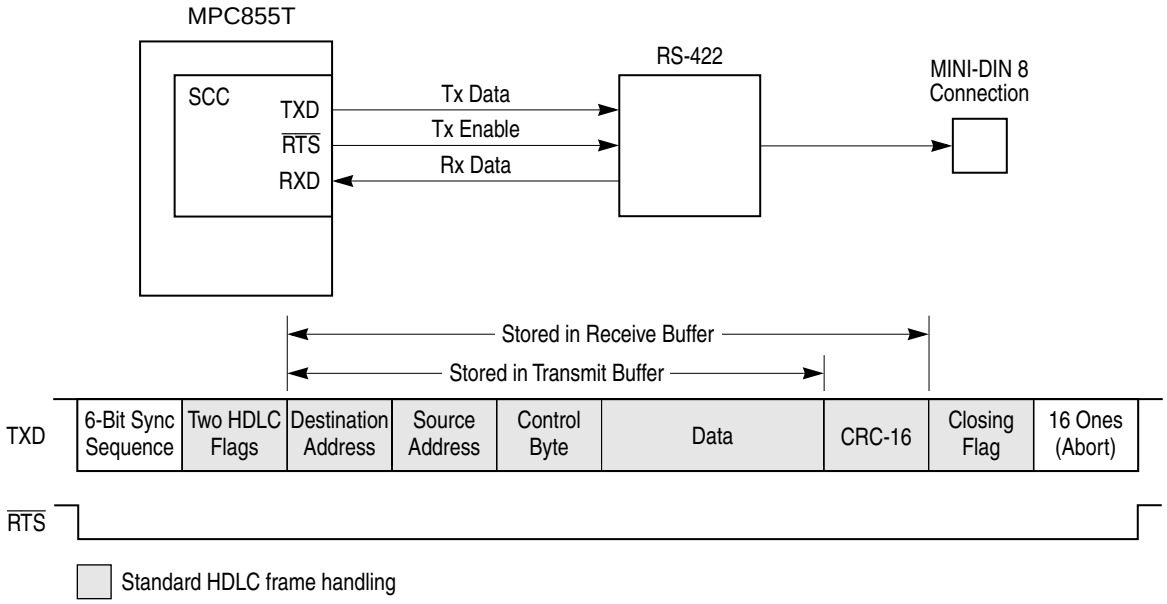


Figure 24-2. Connecting the MPC855T to LocalTalk

The 16× overspeed of a 3.686-MHz 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 3.686 MHz frequency. The MPC855T asserts $\overline{\text{RTS}}$ throughout the duration of the frame so that $\overline{\text{RTS}}$ can be used to enable the RS-422 transmit driver.

24.4 Programming the SCC in AppleTalk Mode

The AppleTalk controller is implemented by setting certain bits in the HDLC controller. Otherwise, Chapter 23, “SCC HDLC Mode,” describes how to program the HDLC controller. Use GSMR, PSMR, or TODR to program the AppleTalk controller.

24.4.1 Programming the GSMR

Program the GSMR as described below:

1. Set MODE to 0b0010 (AppleTalk).

2. Set DIAG to 0b00 for normal operation, with $\overline{CD}$ and $\overline{CTS}$ grounded or configured for parallel I/O. This causes $\overline{CD}$ and $\overline{CTS}$ to be internally asserted to the SCC.
3. Set RDCR and TDCR to (0b10) a 16× clock.
4. Set the TENC and RENC bits to 0b010 (FM0).
5. Clear TEND for default operation.
6. Set TPP to 0b11 for a preamble pattern of all ones.
7. Set TPL to 0b000 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. These bits may be modified on-the-fly if the AppleTalk protocol is selected.
8. Clear TINV and RINV so data will not be inverted.
9. Set TSNC to 1.5 bit times (0b10).
10. Clear EDGE. Both the positive and negative edges are used to change the sample point (default).
11. Clear RTSM (default).
12. Set all other bits to zero or default.
13. Set ENT and ENR as the last step to begin operation.

24.4.2 Programming the PSMR

Follow these steps to program the protocol-specific mode register:

1. Set NOF to 0b0001 giving two flags before frames (one opening flag, plus one additional flag).
2. Set CRC 16-bit CRC-CCITT.
3. Set DRT.
4. Set all other bits to zero or default.

For the PSMR definition, see Section 23.8, “HDLC Mode Register (PSMR).”

24.4.3 Programming the TODR

Use the transmit-on-demand (TODR) register to expedite a transmit frame. See Section 21.2.4, “Transmit-on-Demand Register (TODR).”

24.4.4 SCC AppleTalk Programming Example

Except for the previously discussed register programming, use the example in Section 23.13.1, “SCC HDLC Programming Example #1.”

Chapter 25
SCC Asynchronous HDLC Mode and IrDA

Asynchronous HDLC and IrDA uses HDLC framing techniques with UART-type characters. This document refers to both protocols collectively as asynchronous HDLC. The asynchronous HDLC protocol is typically used as the physical layer for point-to-point protocol (PPP) and the infrared link access protocol (IrLAP). Although asynchronous HDLC can be implemented in conjunction with the core, it is more efficient and less computationally intensive to let the CPM handle framing and transparency functions.

The RFC 1549 octet stuffing/unstuffing provided by this mode supports only asynchronous transmission. This mode cannot be used to provide octet stuffing for synchronous communication lines.

25.1 Asynchronous HDLC Features

The following list summarizes the main features of the SCC in asynchronous HDLC mode:

- Flexible buffer structure lets all or part of a frame be sent or 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 a single 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 a STOP TRANSMIT command
- Automatic transmission of idle characters between frames and between characters

25.2 Asynchronous HDLC Frame Transmission Processing

The SCC in asynchronous HDLC mode (asynchronous HDLC controller) works with minimal core intervention. When the core enables the transmitter and sets TxBD[R] in the

first BD of the table, the asynchronous HDLC controller fetches data from memory and starts sending the frame. If the current TxBD[L] is set (last buffer of a frame), the CRC and closing flag are appended. If TxBD[CM] is zero, the transmitter updates frame status bits in the BD and clears TxBD[R]. If TxBD[I] is set, the controller sets SCCE[TXB] so an interrupt can be generated after each buffer, after a group of buffers, or after each frame is sent.

If TxBD[CM] is set, the asynchronous HDLC transmitter updates frame status bits in the BD after transmission but does not clear TxBD[R]. The transmitter then proceeds to the next TxBD and if necessary waits until it is ready. As the transmitter sends data, it performs the transparency encoding specified by the protocol. See Section 25.4, “Transmitter Transparency Encoding.”

Figure 25-1. Asynchronous HDLC Frame Structure

BOF	Address	Control	Information	FCS (CRC)	EOF
8 bits	8 bits	8 bits	M * 8 bits	2 • 8 bits	8 bits

To rearrange buffers, such as for error handling or to expedite data ahead of previously linked buffers, issue a STOP TRANSMIT command before modifying the TxBD table or directly changing the current TxBD pointer TBPTR. When the asynchronous HDLC controller receives a STOP TRANSMIT command, it stops the transmission and sends the asynchronous HDLC abort sequence. It then sends idle characters until the RESTART TRANSMIT command is given, at which point it resumes transmission with the next TxBD.

25.3 Asynchronous HDLC Frame Reception Processing

The asynchronous HDLC receiver is designed to work with minimal core intervention. It can decode transparency characters, check the CRC of the frame, and detect errors on the line and in the controller. When the core enables the receiver and the receiver detects a data byte of the incoming frame preceded by one or more opening flags, the asynchronous HDLC controller fetches the next BD. If RxBD[E] is set, the controller starts transferring the incoming frame into the buffer. When the buffer is full, the controller clears RxBD[E]. If the incoming frame is larger than the buffer, the controller fetches the next BD, and if E is set, continues transferring the rest of the frame into its buffer.

The receiver decodes the transparency character required by asynchronous HDLC protocol as described in Section 25.5, “Receiver Transparency Decoding.” When the frame ends, the controller checks the incoming CRC field and writes it to the buffer. The controller then updates RxBD[Data Length] with the total frame length, including the CRC bytes. The controller sets RxBD[L], writes the frame status bits, and clears RxBD[E] (if RxBD[CM] is zero). It then sets SCCE[RXF], which indicates that a frame was received and is in memory. The controller then waits for the start of the next frame, which may or may not have an opening flag.

25.4 Transmitter Transparency Encoding

The asynchronous HDLC transmitter encodes characters according to RFC 1549, a de facto standard of the Internet Engineering Task Force (IETF). It examines outgoing bytes and performs the transparency algorithm for the following conditions:

- The byte is a flag (0x7E for PPP, 0xC0 or 0xC1 for IrLAP)
- The byte is a control-escape character (0x7D)
- The byte value is 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 sent instead of the byte. The sequence consists of the control-escape character (0x7D) followed by the original byte exclusive-ORed with 0x20.

25.5 Receiver Transparency Decoding

The asynchronous HDLC receiver decodes characters according to RFC 1549. To recover the original data, it examines incoming data bytes and performs the transparency algorithm in the following ways:

- It discards characters whose corresponding bit is 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 original frame.
- It reverses the transmission transparency sequence by discarding a received control-escape character (0x7D) and exclusive-ORing the following byte with 0x20 before performing the CRC calculation and writing the byte into memory.

Figure 25-2 shows the algorithm because some cases are not covered by RFC 1549.

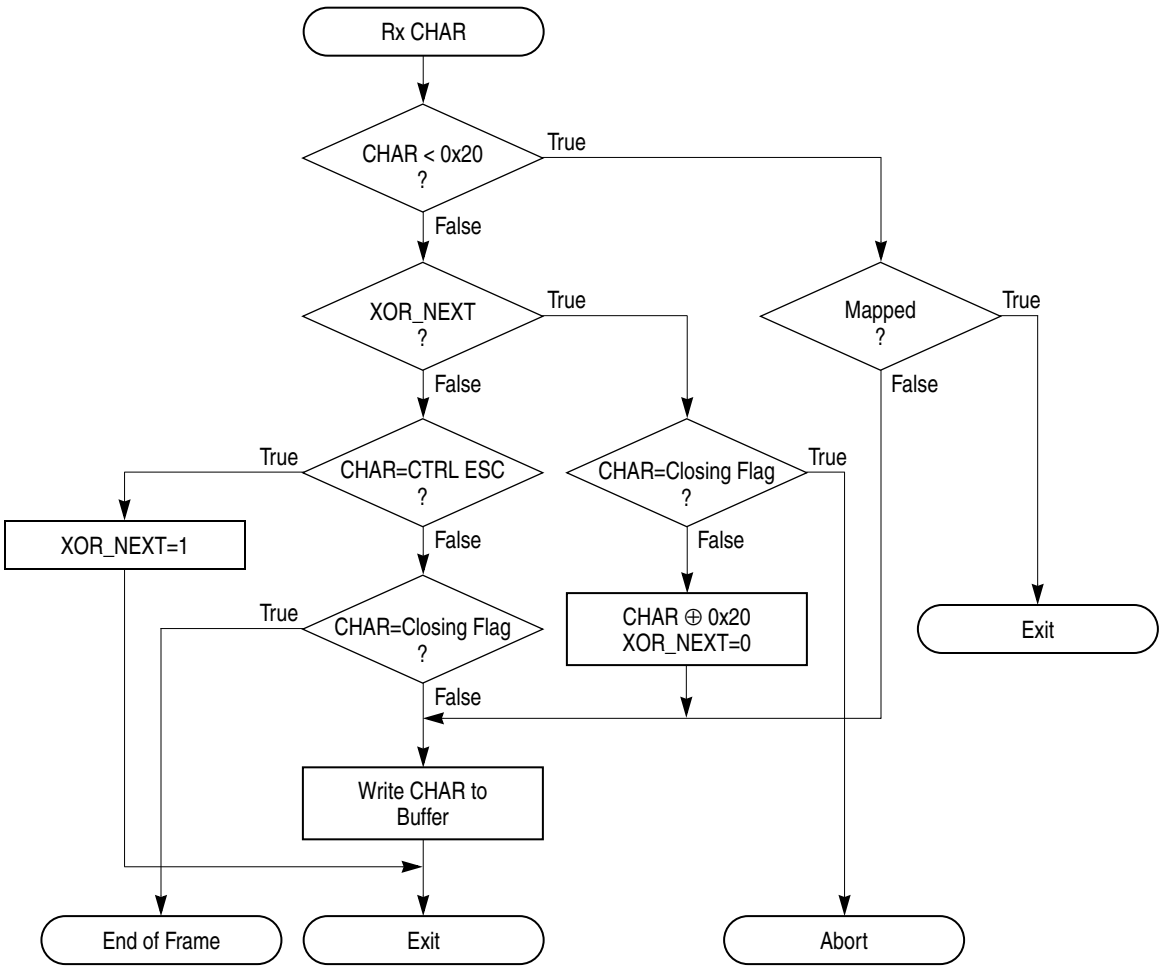


Figure 25-2. Receive Flowchart

25.6 Exceptions to RFC 1549

- An unmapped control character that follows 0x7D is modified by the XOR process. The CRC check should catch this.
- In addition to the abort sequence, frames are terminated by the following errors:
  - — $\overline{\text{CD}}$ (carrier detect) lost
  - — Receiver overrun
  - — Framing error
  - — Break sequence
- If an invalid sequence(0x7D7D) is received, the first control escape character is discarded, and the second is unconditionally XORed with 0x20. The sequence is thus stored in the buffer as 0x5D.

25.7 Asynchronous HDLC Channel Implementation

The following points are specific to asynchronous HDLC channel implementation:

- Flag sequence—The transmitter automatically generates the opening and closing flags. The receiver removes opening and closing flags before writing a frame to memory and receives frames with only one shared flag between frames, ignoring multiple flags.
- Address field—The address field is neither generated nor examined by the microcode while sending or receiving. The destination address field of the frame must be included in the Tx buffer. 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 during a transfer. The control field of the frame must be included in the buffer. Any control field compression, expansion, or checking is done by the core.
- Frame check sequence—When sending, the frame check sequence (FCS) is appended to the frame before the closing flag is sent. The FCS is generated on the original frame before transparency characters, start/stop bits, or flags are added. When receiving, the FCS is checked automatically and calculated after any transparency characters, start/stop bits, and flags are removed. For both, the controller uses only a 16-bit CRC-CCITT polynomial.
- Encoding—The asynchronous HDLC controller supports 8 data bits, one start bit, one stop bit, and no parity. Program PSMR[CHLN] to 0b11 for proper operation.
- Idle characters—When sending, the asynchronous HDLC controller sends idle characters when no data is available; when receiving, it ignores idle characters.

25.8 Asynchronous HDLC Mode Parameter RAM

For asynchronous HDLC mode, the protocol-specific area of the SCC parameter RAM is mapped as in Table 25-1.

Table 25-1. Asynchronous HDLC-Specific SCC Parameter RAM Memory Map

Offset [1]	Name	Width	Description
0x30	—	Word	Reserved
0x34	**C_MASK**	Word	CRC constant. Initialize with 0x0000_F0B8.
0x38	**C_PRES**	Word	CRC preset. Initialize with 0x0000_FFFF.
0x3C	**BOF**	Hword	Beginning-of-flag-character. Initialize to PPP-0x7E, IrLAP - 0xC0.
0x3E	**EOF**	Hword	End-of-flag character. Initialize to PPP-0x7E, IrLAP-0xC1.
0x40	**ESC**	Hword	Control escape character. Initialize to 0x7D for both PPP and IrLAP.
0x42	—	Word	Reserved

Table 25-1. Asynchronous HDLC-Specific SCC Parameter RAM Memory Map (continued)

Offset [1]	Name	Width	Description
0x46	**ZERO**	Hword	Clear this field.
0x48	—	Hword	Reserved
0x4A	**RFTHR**	Hword	Received frames threshold. Number of Rx frames needed to trigger SCCE[RXF]
0x4C	—	Word	Reserved
0x50	**TXCTL_TBL**	Word	Control character tables. Stores the bit array used for the Tx/Rx control characters. See Figure 25-3. Each bit corresponds to a character that should be mapped according to RFC 1549. If a TXCTL_TBL bit is set, its corresponding character is mapped; otherwise, it is not mapped. If an RXCTL_TBL bit is set, its corresponding character is discarded if received; otherwise, it is received normally. TXCTL_TBL and RXCTL_TBL should be initialized to zero for IrLAP.
0x54	**RXCTL_TBL**	Word	
0x58	**NOF**	Hword	Number of opening flags to be sent at the beginning of a frame. A value of n corresponds to n+1 flags.
0x5A	—	Hword	Reserved

[1] From SCC base. SCC base = IMMR + 0x3C00 (SCC1)

Figure 25-3 shows bit arrangements for TXCTL_TBL and RXCTL_TBL.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x1F	0x1E	0x1D	0x1C	0x1B	0x1A	0x19	0x18	0x17	0x16	0x15	0x14	0x13	0x12	0x11	0x10

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0x0F	0x0E	0x0D	0x0C	0x0B	0x0A	0x09	0x08	0x07	0x06	0x05	0x04	0x03	0x02	0x01	0x00

Figure 25-3. TXCTL_TBL/RXCTL_TBL

25.9 Configuring GSMR and DSR for Asynchronous HDLC

General SCC parameters can be configured as described in Chapter 21, “Serial Communications Controller,” except for the following changes to the general SCC mode register and the data synchronization register.

25.9.1 General SCC Mode Register (GSMR)

Table 25-2 shows asynchronous HDLC-specific information for the GSMR.

Table 25-2. Asynchronous HDLC-Specific GSMR Field Descriptions

Name	Description
RFW	Rx FIFO width (GSMR_H[26]) 0 Do not use. 1 Low-latency operation—for character-oriented protocols like UART, BISYNC, and asynchronous HDLC. The Rx FIFO is 8 bits wide and the Rx FIFO is one-fourth its normal size (8 bytes for SCC1). This allows each character to be written to the buffer without waiting for 32 bits to be received.
TDCR/ RDCR	Tx/Rx divide clock rate (GSMR_L[14–15/16–17]). For asynchronous HDLC mode, 8×, 16×, or 32× must be chosen. Set TDCR = RDCR 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).

25.9.2 Data Synchronization Register (DSR)

The data synchronization register (DSR) is reserved in asynchronous HDLC mode. It should be left in its reset state of 0x7E7E.

25.10 Programming the Asynchronous HDLC Controller

Asynchronous HDLC mode is selected for an SCC by writing GSMR_L[MODE] = 0b0110. The asynchronous HDLC controller uses the same buffer and BD data structure as other modes and supports multibuffer operation. Receive errors are reported through the RxBD; transmit errors are reported through the TxBD. Status line information ($\overline{CD}$ and $\overline{CTS}$) is reported through the port C pins; a maskable interrupt is generated when the status of either line changes.

25.11 Asynchronous HDLC Commands

The transmit and receive commands are issued to the CP command register (CPCR).

Transmit commands are described in Table 25-3. After a hardware or software reset and a channel is enabled in the GSMR, the transmitter starts polling the first BD in the TxBD table every 8 transmit clocks, or immediately if TODR[TOD] = 1, and begins sending data if TxBD[R] is set.

Table 25-3. Transmit Commands

Command	Description
STOP TRANSMIT	Sends the asynchronous HDLC abort sequence (0x7D;0x7E for PPP, 0x7D; 0xC1 for IrLAP) and disables data transmission. If the asynchronous HDLC controller receives this command during frame transmission, the abort sequence is put in the FIFO and the transmitter does not try to send more data from the current BD or advance to the next TxBD. The BD to be terminated is indicated by the TBPTR entry in the parameter RAM table. Note that unlike with other SCC protocols, the STOP TRANSMIT command does not flush the FIFO. Up to 32 characters can be sent ahead of the abort sequence unless GSMR_H[TFL] = 1.
GRACEFUL STOP TRANSMIT	Not supported by the asynchronous HDLC controller.
RESTART TRANSMIT	Reenables transmission of characters; the asynchronous HDLC controller expects it after a STOP TRANSMIT command or transmitter error. The controller continues sending from the first character in the buffer using the current TxBD (pointed to by TBPTR).
INIT TX PARAMETERS	Initializes all Tx parameters in this channel's parameter RAM to reset state. It must be issued only when the transmitter is disabled. The INIT TX AND RX PARAMETERS command resets both Tx and Rx parameters.

Table 25-4 describes receive commands. After hardware or software is reset and a channel is enabled in the GSMR, reception begins with the first BD in the RxBD table.

Table 25-4. Receive Commands

Command	Description
ENTER HUNT MODE	Forces the asynchronous HDLC controller to close the current RxBD, if it is in use, and enter hunt mode. Reception resumes after the controller finds a frame preceded by one or more opening flags.
CLOSE RXBD	Not supported by the asynchronous HDLC controller.
INIT RX PARAMETERS	Initializes all Rx parameters in the channel's parameter RAM to reset state. Issue only when the receiver is disabled. The INIT TX AND RX PARAMETERS command resets both Tx and Rx parameters.

25.12 Handling Errors in the Asynchronous HDLC Controller

The asynchronous HDLC controller reports frame reception and transmission error conditions using the channel BDs and the asynchronous HDLC event register (SCCE). Table 25-5 describes transmit errors.

Table 25-5. Transmit Errors

Error	Description
$\overline{\text{CTS}}$ Lost during Frame Transmission	The channel stops sending the buffer, closes it, sets SCCE[TXE] and TxBD[CT]. The channel resumes sending from the next TxBD after a RESTART TRANSMIT command is issued.

Table 25-6 describes reception errors.

Table 25-6. Receive Errors

Error	Description
Overrun	SCC1 has 32-byte Rx FIFOs. Overrun occurs when the CP cannot keep up with the data rate or the SDMA channel cannot write the received data to memory. The previous data byte and frame status are lost. The controller closes the buffer and sets RxBD[OV] and SCCE[RXF]. The receiver then looks for the next frame.
$\overline{CD}$ Lost during Frame Reception	The channel stops receiving frames, closes the buffer, and sets SCCE[RXF] and RxBD[CD]. This error has 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 $\overline{CD}$ is reasserted.
Abort Sequence	When an abort sequence (0x7D, 0x7E for PPP; 0x7D, 0xC1 for IrLAP) is detected, the channel closes the buffer by setting SCCE[RXF] and RxBD[AB]. CRC error status is not checked on aborted frames. If no frame is being received, the next BD is opened and then closed with RxBD[AB] set.
CRC	The channel writes the received cyclic redundancy check to the buffer, closes the buffer, and sets SCCE[RXF] and RxBD[CR]. After receiving this error, the receiver prepares to receive the next frame.
Break Sequence Received	The receiver detected the first character in a break sequence. The channel closes the buffer and sets SCCE[RXF] and RxBD[BRK]. CRC error status is not checked. SCCE[BRKS] is set when the first break of a sequence is found; SCCE[BRKE] is set when an idle bit is received after a break sequence.

25.13 SCC Asynchronous HDLC Registers

The following sections describe the SCC registers when in asynchronous HDLC mode.

25.13.1 Asynchronous HDLC Event Register (SCCE)/Asynchronous HDLC Mask Register (SCCM)

The SCC event register (SCCE) is used as the asynchronous HDLC event register to generate interrupts and report events recognized by the asynchronous HDLC channel. When an event is recognized, the asynchronous HDLC controller sets the corresponding SCCE bit. Interrupts can be masked by clearing the appropriate bit in the asynchronous HDLC mask register (SCCM). SCCE bits, shown in Figure 25-4, are cleared by writing ones—writing zeros has no effect. Unmasked SCCE bits must be cleared before the CPM clears the internal interrupt request.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—			GLR	GLT	—		IDL	—	BRKE	BRKS	TXE	RXF	BSY	TXB	RXB
Reset	0															
R/W	R/W															
Addr	0xA10 (SCCE1)/0xA14 (SCCM1)															

Figure 25-4. Asynchronous HDLC Event Register (SCCE)/Asynchronous HDLC Mask Register (SCCM)

Table 25-7 describes SCCE/SCCM fields.

Table 25-7. SCCE/SCCM Field Descriptions

Bits	Name	Description
0–2	—	Reserved, should be cleared.
3	GLR	Glitch on Rx. Set when the SCC finds a Rx clock glitch.
4	GLT	Glitch on Tx. Set when the SCC finds a Tx clock glitch.
5–6	—	Reserved, should be cleared.
7	IDL	Idle sequence status changed. Set when serial line status changes. Real-time status can be read in SCCS[ID].
8	—	Reserved, should be cleared.
9	TXE	Tx error. Set when an error occurs on the transmitter channel.
10	BRKE	Break end. Marks the end of a break sequence—set when an idle bit is detected after a break sequence.
11	BRKS	Break start. Set when the first break character of a break sequence is received. Only one BRKS event occurs per break sequence, no matter the length of the sequence.
12	RXF	Rx frame. Set when the number of frames specified in RFTHR are received. RXF is set no sooner than when the midpoint of the closing flag's stop bit arrives.
13	BSY	Busy condition. Set when a frame is received and discarded due to a buffer shortage.
14	TXB	Transmit buffer. Set when a buffer with TxBD[I] set is sent on the channel, not before the last bit of the closing flag begins its transmission if the buffer is the last one in the frame. Otherwise, TXB is set after the last byte of the buffer is written to the Tx FIFO.
15	RXB	Rx buffer. Set when a buffer with RxBD[I] set and RxBD[L] cleared is received over the channel.

25.13.2SCC Asynchronous HDLC Status Register (SCCS)

The SCC asynchronous HDLC status register (SCCS), shown in Figure 25-5, monitors the real-time status of RXD. The real-time status of $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ is part of the port C parallel I/O.

Bit	0	1	2	3	4	5	6	7
Field	—							ID
Reset	0000_0000_0000_0000							
R/W	R							
Addr	0xA17 (SCCS1)							

Figure 25-5. SCC Status Register for Asynchronous HDLC Mode (SCCS)

Table 25-8 describes asynchronous HDLC SCCS fields.

Table 25-8. Asynchronous HDLC SCCS Field Descriptions

Bits	Name	Description
0–6	—	Reserved, should be cleared.
7	ID	Idle status. Set when RXD has been a logic one for at least a full character time. 0 The line is not idle. 1 The line is idle.

25.13.3 Asynchronous HDLC Mode Register (PSMR)

When the SCC is in asynchronous HDLC mode, the PSMR, shown in Figure 25-6, acts as the asynchronous HDLC mode register.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	FLC	—	CHLN		—											
Reset	0															
R/w	R/W															
Addr	0xA08 (PSMR1)															

Figure 25-6. Asynchronous HDLC Mode Register (PSMR)

Table 25-9 describes PSMR fields.

Table 25-9. PSMR Field Descriptions

Bits	Name	Description
0	FLC	Flow control 0 Normal operation. 1 Asynchronous flow control. When $\overline{\text{CTS}}$ is negated, the transmitter stops at the end of the current character. If $\overline{\text{CTS}}$ remains negated past the middle of the character, the next full character is sent before transmission stops. If $\overline{\text{CTS}}$ is reasserted, transmission resumes from where it stopped and no $\overline{\text{CTS}}$ lost error is reported. Only idle characters are sent while $\overline{\text{CTS}}$ is negated.
1	—	Reserved, should be cleared.
2–3	CHLN	Character length. On other protocols CHLN is the number of data bits in a character. For asynchronous HDLC mode and IrDA modes, CHLN must be set to 0b11 (indicating a character length of 8 bits).
4–15	—	Reserved, should be cleared.

25.14 SCC Asynchronous HDLC RxBDs

The CPM uses the RxBD, shown in Figure 25-7, to report on received data. An example of the RxBD process is shown in Figure 23-5 of Section 23.9, “SCC HDLC Receive Buffer Descriptor (RxBD).”

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**E**	—	**W**	**I**	L	F	**CM**	—	BRK	BOF	—		AB	CR	OV	CD
Offset + 2	Data Length															
Offset + 4	**Rx Buffer Pointer**															
Offset + 6																

Figure 25-7. SCC Asynchronous HDLC RxBDs

Table 25-10 describes the SCC asynchronous HDLC RxBD status and control fields.

Table 25-10. Asynchronous HDLC RxBD Status and Control Field Descriptions

Bits	Name	Description
0	**E**	Empty. 0 The buffer is full or stops receiving because of an error. The core can read or update any fields of this RxBD. The CPM cannot reuse this BD while E = 0. 1 The buffer is not full. The CP controls the BD and buffer. The core should not update the BD.
1	—	Reserved, should be cleared.
2	**W**	Wrap (last BD in table). 0 Not the last BD in the table. 1 Last BD in the table. After this buffer is used, the CPM receives incoming data using the BD pointed to by RBASE. The number of RxBDs in a table is determined by the W bit.
3	**I**	Interrupt. 0 SCCE[RXB] is not set after this buffer is used. SCCE[RXF] is unaffected. 1 SCCE[RXB] or SCCE[RXF] is set when this buffer is used by the asynchronous HDLC controller.
4	L	Last in frame. 0 Not the last buffer in a frame. 1 Set by SCC when a buffer is the last in a frame which happens when a closing flag or error is received. If an error occurs, one or more of the BRK, CD, OV, BOF, CR, and AB bits are set. The SCC updates RxBD[Data Length].
5	F	First in frame. Set by the SCC when this buffer is the first in a frame. 0 Not the first buffer in a frame. 1 First buffer in a frame.
6	**CM**	Continuous mode. 0 Normal operation. 1 The CP does not clear E after the BD is closed allowing a buffer to be overwritten when the CP next accesses the BD. However, E is cleared if an error other than CRC occurs during reception, regardless of CM.
7	—	Reserved, should be cleared.
8	BRK	Break character received. Set when a frame is closed because a break character is received.

Table 25-10. Asynchronous HDLC RxBD Status and Control Field Descriptions (continued)

Bits	Name	Description
9	BOF	Beginning of frame. Set when a frame is closed because a BOF character is received instead of the expected EOF.
10–11	—	Reserved, should be cleared.
12	AB	Rx abort sequence. Set when an abort sequence or framing error terminates a frame.
13	CR	Rx CRC error. Set when a frame has a CRC error. Received CRC bytes are written to the buffer.
14	OV	Overrun. Set when a receiver overrun occurs during frame reception.
15	CD	Carrier detect lost. Set when $\overline{CD}$ is negated during frame reception.

The data length and buffer pointer fields are described in Section 21.3, “SCC Buffer Descriptors (BDs).” Because asynchronous HDLC is a frame-based protocol, RxBD[Data Length] of the last buffer of a frame contains the total number of frame bytes, including the 2 or 4 bytes for CRC.

25.15 SCC Asynchronous HDLC TxBDs

The CPM uses the TxBD, shown in Figure 25-8, to confirm transmissions and indicate error conditions.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**R**	—	**W**	**I**	**L**	—	**CM**	—								CT
Offset + 2	**Data Length**															
Offset + 4	**Tx Buffer Pointer**															
Offset + 6																

Figure 25-8. SCC Asynchronous HDLC TxBDs

Table 25-11 describes the SCC asynchronous HDLC TxBD status and control fields.

Table 25-11. Asynchronous HDLC TxBD Status and Control Field Descriptions

Bits	Name	Description
0	**R**	Ready. 0 The buffer is not ready for transmission; the BD and the buffer can be updated. The CPM clears R after the buffer is sent or after an error condition. 1 The buffer is ready but is not sent or is being sent. Do not update the BD while R =1.
1	—	Reserved, should be cleared.

Table 25-11. Asynchronous HDLC TxBD Status and Control Field Descriptions (continued)

Bits	Name	Description
2	**W**	Wrap (last BD in table). 0 Not the last BD in the table. 1 The last BD in the table. After this buffer is used, the CPM sends incoming data using the BD pointed to by TBASE. The number of TxBDs in this table are determined only by the W bit.
3	**I**	Interrupt. 0 SCCE[TXB] is not set after this buffer is sent. 1 SCCE[TXB] is set when this buffer is sent by the asynchronous HDLC controller.
4	**L**	Last. 0 Not the last buffer in the current frame. 1 Last buffer in the current frame. The proper CRC and closing flag are sent after the last byte.
5	—	Reserved, should be cleared.
6	**CM**	Continuous mode. 0 Normal operation. 1 The CP does not clear R after this BD is closed, allowing its buffer to be resent when the CP next accesses this BD. However, R is cleared if an error occurs during transmission, regardless of CM.
7–14	—	Reserved, should be cleared.
15	CT	$\overline{\text{CTS}}$ lost. In NMSI mode, $\overline{\text{CTS}}$ is lost during frame transmission. If more than one buffer has data in the FIFO when this error occurs, CT is set in the currently open TxBD. Written by the asynchronous HDLC controller after it finishes sending the buffer.

The data length and buffer pointer fields are described in Section 21.3, “SCC Buffer Descriptors (BDs).”

25.16 Differences between HDLC and Asynchronous HDLC

The basic differences between HDLC and asynchronous HDLC modes are as follows:

- Asynchronous HDLC does not support the GRACEFUL STOP TRANSMIT command.
- Because asynchronous HDLC has no maximum received frame length counter, it receives all characters between opening and closing flags. There is no way to keep it from writing to memory. This does not affect the number of bytes received into a specific BD. A frame over the maximum length 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 $\overline{\text{CD}}$ lost error occurs, the frame is closed and the partial character is not written to memory. Thus, the octet count reflects only the number of bytes written to memory.
- The automatic error counters in the HDLC controller are not implemented in the asynchronous HDLC controller.

- Noisy characters (characters for which all three samples are not identical) are not accounted for in the asynchronous HDLC controller. It is assumed that the CRC catches any data integrity problems.

25.17 SCC Asynchronous HDLC Programming Example

The following example shows initialization for an SCC in asynchronous HDLC mode.

1. Initialize SDCR.
2. In NMSI mode, configure ports A and C to enable RXD, TXD, $\overline{\text{CTS}}$, $\overline{\text{CD}}$, and $\overline{\text{RTS}}$. In other modes, configure the TSA and its pins.
3. Configure a baud rate generator to the appropriate channel clocking frequency.
4. Program SICR. Route the BRG clocking to the SCC and select whether the channel is using the TSA or the NMSI.
5. Point RBASE and TBASE in the SCC parameter RAM to the first RxBD and TxBD.
6. Issue the INIT RX AND TX PARAMETERS command for the SCC.
7. Program RFCR and TFCR.
8. Write MRBLR with the maximum Rx buffer size.
9. Write C_MASK and C_PRES with the standard values.
10. Clear the Zero register.
11. Program RFTHR to the number of frames to be received before generating an interrupt.
12. Program the control character tables, TXCTL_TBL and RXCTL_TBL.
13. Initialize all RxBDs.
14. Initialize all TxBDs.
15. Clear SCCE by writing 0xFFFF to it.
16. Program SCCM to enable all preferred interrupts.
17. Program GSMR_H.
18. Program GSMR_L to asynchronous HDLC mode, but do not turn on the transmitter or receiver.
19. Set the PSMR appropriately. See Section 25.13.3, “Asynchronous HDLC Mode Register (PSMR).”
20. Enable the transmitter and receiver in GSMR_L.

Chapter 26
SCC BISYNC Mode

The byte-oriented BISYNC protocol was developed by IBM for use in networking products. There are three classes of BISYNC frames—transparent, nontransparent with header, and nontransparent without header, shown in Figure 26-1. The transparent frame type in BISYNC is not related to transparent mode, discussed in Chapter 28, “SCC Transparent Mode.” Transparent BISYNC mode allows full binary data to be sent with any possible character pattern. Each class of frame starts with a standard two-octet synchronization pattern and ends with a block check code (BCC). The end-of-text character (ETX) is used to separate the text and BCC fields.

Nontransparent with Header

SYN1	SYN2	SOH	Header	STX	Text	ETX	BCC

Nontransparent without Header

SYN1	SYN2	STX	Text	ETX	BCC

Transparent

SYN1	SYN2	DLE	STX	Transparent Text	DLE	ETX	BCC

Figure 26-1. Classes of BISYNC Frames

The bulk of a frame is divided into fields whose meaning depends on the frame type. The BCC is a 16-bit CRC format if 8-bit characters are used; it is a combination longitudinal (sum check) and vertical (parity) redundancy check if 7-bit characters are used. In transparent operation, a special character (DLE) is defined that tells the receiver that the next character is text, allowing BISYNC control characters to be valid text data in a frame. A DLE sent as data must be preceded by a DLE character. This is sometimes called byte-stuffing. The physical layer of the BISYNC communications link must synchronize the receiver and transmitter, usually by sending at least one pair of synchronization characters before each frame.

BISYNC protocol is unusual in that a transmit underrun need not be an error. If an underrun occurs, a synchronization pattern is sent until data is again ready. In nontransparent operation, the receiver discards additional synchronization characters (SYNCs) as they are

received. In transparent mode, DLE-SYNC pairs are discarded. Normally, for proper transmission, an underrun must not occur between the DLE and its following character. This failure mode cannot occur with the MPC855T.

The SCC can be configured as a BISYNC controller to handle basic BISYNC protocol in normal and transparent modes. The controller can work with the time-slot assigner (TSA) or nonmultiplexed serial interface (NMSI). The SCC supports modem lines by connecting to port C pins or general-purpose I/O pins. The controller has separate transmit and receive sections whose operations are asynchronous with the core.

26.1 Features

The following list summarizes features of the SCC in BISYNC mode:

- Flexible data buffers
- Eight control character recognition registers
- Automatic SYNC1–SYNC2 detection
- 16-bit pattern (bisync)
- 8-bit pattern (monosync)
- 4-bit pattern (nibblesync)
- External SYNC pin support
- SYNC/DLE stripping and insertion
- CRC16 and LRC (sum check) generation/checking
- VRC (parity) generation/checking
- Supports BISYNC transparent operation
- Maintains parity error counter
- Reverse data mode capability

26.2 SCC BISYNC Channel Frame Transmission

The BISYNC transmitter is designed to work with almost no core intervention. When the transmitter is enabled, it starts sending SYN1–SYN2 pairs in the data synchronization register (DSR) or idles as programmed in the PSMR. The BISYNC controller polls the first BD in the channel's TxBD table. If there is a message to send, the controller fetches the message from memory and starts sending it after the SYN1–SYN2 pair. The entire pair is always sent, regardless of GSMR[SYNL].

After a buffer is sent, if the last (TxBD[L]) and the Tx block check sequence (TxBD[TB]) bits are set, the BISYNC controller appends the CRC16/LRC and then writes the message status bits in TxBD status and control fields and clears the ready bit, TxBD[R]. It then starts sending the SYN1–SYN2 pairs or idles, according to GSMR[RTSM]. If the end of the current BD is reached and TxBD[L] is not set, only TxBD[R] is cleared. In both cases, an

interrupt is issued according to TxBD[I]. TxBD[I] controls whether interrupts are generated after transmission of each buffer, a specific buffer, or each block. The controller then proceeds to the next BD.

If no additional buffers have been sent to the controller for transmission, an in-frame underrun is detected and the controller starts sending syncs or idles. If the controller is in transparent mode, it sends DLE-sync pairs. Characters are included in the block check sequence (BCS) calculation on a per-buffer basis. Each buffer can be programmed independently to be included or excluded from the BCS calculation; thus, excluded characters must reside in a separate buffer. The controller can reset the BCS generator before sending a specific buffer. In transparent mode, the controller inserts a DLE before sending a DLE character, so that only one DLE is used in the calculation.

26.3 SCC BISYNC Channel Frame Reception

Although the receiver is designed to work with almost no core intervention, the user can intervene on a per-byte basis if necessary. The receiver performs CRC16, longitudinal (LRC) or vertical redundancy (VRC) checking, sync stripping in normal mode, DLE-sync stripping, stripping of the first DLE in DLE-DLE pairs in transparent mode, and control character recognition. Control characters are discussed in Section 26.6, “SCC BISYNC Control Character Recognition.”

When enabled, the receiver enters hunt mode where the data is shifted into the receiver shift register one bit at a time and the contents of the shift register are compared to the contents of DSR[SYN1, SYN2]. If the two are unequal, the next bit is shifted in and the comparison is repeated. When registers match, hunt mode is terminated and character assembly begins. The controller is character-synchronized and performs SYNC stripping and message reception. It reverts to hunt mode when it receives an ENTER HUNT MODE command, an error condition, or an appropriate control character.

When receiving data, the controller updates the BCS bit in the BD for each byte transferred. When the buffer is full, the controller clears the E bit in the BD and generates an interrupt if the I bit in the BD is set. If incoming data exceeds the buffer length, the controller fetches the next BD; if E is zero, reception continues to its buffer.

When a BCS is received, it is checked and written to the buffer. The BISYNC controller sets the last bit, writes the message status bits into the BD, clears the E bit, and then generates a maskable interrupt, indicating that a block of data was received and is in memory. The BCS calculations do not include SYNCs (in nontransparent mode) or DLE-SYNC pairs (in transparent mode).

Note that GSMR_H[RFW] should be set for an 8-bit-wide receive FIFO for the BISYNC receiver. See Section 21.2.1, “General SCC Mode Register (GSMR).”

26.4 SCC BISYNC Parameter RAM

When BISYNC mode is selected in GSMR_L[MODE], the protocol-specific area of the SCC parameter RAM is mapped as in Table 26-1.

Table 26-1. SCC BISYNC Parameter RAM Memory Map

Offset [1]	Name	Width	Description
0x30	—	Word	Reserved
0x34	**CRCC**	Word	CRC constant temporary value.
0x38	**PRCRC**	Hword	Preset receiver/transmitter CRC16/LRC. These values should be preset to all ones or zeros, depending on the BCS used.
0x3A	**PTCRC**	Hword	
0x3C	**PAREC**	Hword	Receive parity error counter. This 16-bit (modulo 2^{16}) counter maintained by the CP counts parity errors on receive if the parity feature of BISYNC is enabled. Initialize PAREC while the channel is disabled.
0x3E	**BSYNC**	Hword	BISYNC SYNC register. Contains the value of the SYNC to be sent as the second byte of a DLE–SYNC pair in an underrun condition and stripped from incoming data on receive once the receiver synchronizes to the data using the DSR and SYN1–SYN2 pair. See Section 26.7, “BISYNC SYNC Register (BSYNC).”
0x40	**BDLE**	Hword	BISYNC DLE register. Contains the value to be sent as the first byte of a DLE–SYNC pair and stripped on receive. See Section 26.8, “SCC BISYNC DLE Register (BDLE).”
0x42	**CHARACTER1**	Hword	Control character 1–8. These values represent control characters that the BISYNC controller recognizes. See Section 26.6, “SCC BISYNC Control Character Recognition.”
0x44	**CHARACTER2**	Hword	
0x46	**CHARACTER3**	Hword	
0x48	**CHARACTER4**	Hword	
0x4A	**CHARACTER5**	Hword	
0x4C	**CHARACTER6**	Hword	
0x4E	**CHARACTER7**	Hword	
0x50	**CHARACTER8**	Hword	
0x52	**RCCM**	Hword	Receive control character mask. Masks CHARACTER*n* comparison so control character classes can be defined. Setting a bit enables and clearing a bit masks comparison. See Section 26.6, “SCC BISYNC Control Character Recognition.”

1 From SCC base. SCC base = IMMR + 0x3C00 (SCC1)

The SYN1–SYN2 synchronization characters are programmed in the DSR (see Section 21.2.3, “Data Synchronization Register (DSR).”) The BISYNC controller uses the same basic data structure as other modes; receive and transmit errors are reported through their respective BDs. Line status is reflected on port C pins and a maskable interrupt is generated when the status changes. There are two basic ways to handle BISYNC channels:

- The controller can inspect data on a per-byte basis and interrupt the core each time a byte is received.

- The controller can be programmed so software handles the first two or three bytes. The controller directly handles subsequent data without interrupting the core.

26.5 SCC BISYNC Commands

Transmit and receive commands are issued to the CP command register (CPCR). Transmit commands are described in Table 26-2.

Table 26-2. Transmit Commands

Command	Description
STOP TRANSMIT	After hardware or software is reset and the channel is enabled in the GSMR, the channel is in transmit enable mode and starts polling the first BD every 64 transmit clocks. This command stops transmission after a maximum of 64 additional bits without waiting for the end of the buffer and the transmit FIFO to be flushed. TBPTR is not advanced, no new BD is accessed, and no new buffers are sent for this channel. SYNC–SYNC or DLE–SYNC pairs are sent continually until a RESTART TRANSMIT is issued. A STOP TRANSMIT can be used when an EOT sequence should be sent and transmission should stop. After transmission resumes, the EOT sequence should be the first buffer sent to the controller. Note that the controller remains in transparent or normal mode after it receives a STOP TRANSMIT or RESTART TRANSMIT command.
GRACEFUL STOP TRANSMIT	Stops transmission after the current frame finishes sending or immediately if there is no frame being sent. SCCE[GRA] is set once transmission stops. Then BISYNC transmit parameters and TxBDs can be modified. The TBPTR points to the next TxBD. Transmission resumes when the R bit of the next BD is set and a RESTART TRANSMIT is issued.
RESTART TRANSMIT	Lets characters be sent on the transmit channel. The BISYNC controller expects it after a STOP TRANSMIT or a GRACEFUL STOP TRANSMIT command is issued, after a transmitter error occurs, or after a STOP TRANSMIT is issued and the channel is disabled in its SCCM. The controller resumes transmission from the current TBPTR in the channel's TxBD table.
INIT TX PARAMETERS	Initializes all transmit parameters in the serial channel's parameter RAM to their reset state. Issue only when the transmitter is disabled. INIT TX AND RX PARAMETERS resets transmit and receive parameters.

Receive commands are described in Table 26-3.

Table 26-3. Receive Commands

Command	Description
RESET BCS CALCULATION	Immediately resets the receive BCS accumulator. It can be used to reset the BCS after recognizing a control character, thus signifying that a new block is beginning.
ENTER HUNT MODE	After hardware or software is reset and the channel is enabled in SCCM, the channel is in receive enable mode and uses the first BD. This command forces the controller to stop receiving and enter hunt mode, during which the controller continually scans the data stream for an SYN1–SYN2 sequence as programmed in the DSR. After receiving the command, the current receive buffer is closed and the BCS is reset. Message reception continues using the next BD.
CLOSE RXBD	Used to force the SCC to close the current RxBD if it is in use and to use the next BD for subsequent data. If data is not being received, no action is taken.
INIT RX PARAMETERS	Initializes receive parameters in this serial channel's parameter RAM to reset state. Issue only when the receiver is disabled. An INIT TX AND RX PARAMETERS resets transmit and receive parameters.

26.6 SCC BISYNC Control Character Recognition

The BISYNC controller recognizes special control characters that customize the protocol implemented by the BISYNC controller and aid its operation in a DMA-oriented environment. They are used for receive buffers longer than one byte. In single-byte buffers, each byte can easily be inspected so control character recognition should be disabled.

The control character table lets the BISYNC controller recognize the end of the current block. Because the controller imposes no restrictions on the format of BISYNC blocks, software must respond to received characters and inform the controller of mode changes and of certain protocol events, such as resetting the BCS. Using the control character table correctly allows the remainder of the block to be received without interrupting software.

Up to eight control characters can be defined to inform the BISYNC controller that the end of the current block is reached and whether a BCS is expected after the character. For example, the end-of-text character (ETX) implies an end-of-block (ETB) with a subsequent BCS. An enquiry (ENQ) character designates an end of block without a subsequent BCS. All the control characters are written into the data buffer. The BISYNC controller uses a table of 16-bit entries to support control character recognition. Each entry consists of the control character, an end-of-table bit (E), a BCS expected bit (B), and a hunt mode bit (H). The RCCM entry defines classes of control characters that support masking option.

Offset from SCC Base	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x42	E	B	H	—					CHARACTER1							
0x44	E	B	H	—					CHARACTER2							
0x46	E	B	H	—					CHARACTER3							
0x48	E	B	H	—					CHARACTER4							
0x4A	E	B	H	—					CHARACTER5							
0x4D	E	B	H	—					CHARACTER6							
0x4E	E	B	H	—					CHARACTER7							
0x50	E	B	H	—					CHARACTER8							
0x52	1	1	1	—					MASK VALUE(RCCM)							

Figure 26-2. Control Character Table and RCCM

Table 26-4 describes control character table and RCCM fields.

Table 26-4. Control Character Table and RCCM Field Descriptions

Offset	Bit	Name	Description
0x42–0x50	0	E	End of table. 0 This entry is valid. The lower eight bits are checked against the incoming character. In tables with eight control characters, E should be zero in all eight positions. 1 The entry is not valid. No other valid entries exist beyond this entry.
	1	B	BCS expected. A maskable interrupt is generated after the buffer is closed. 0 The character is written into the receive buffer and the buffer is closed. 1 The character is written into the receive buffer. The receiver waits for one LRC or two CRC bytes of BCS and then closes the buffer. This should be used for ETB, ETX, and ITB.
	2	H	Hunt mode. Enables hunt mode when the current buffer is closed. 0 The BISYNC controller maintains character synchronization after closing this buffer. 1 The BISYNC controller enters hunt mode after closing the buffer. When the B bit is set, the controller enters hunt mode after receiving the BCS.
	3–7	—	Reserved
	8–15	CHARACTER*n*	Control character 1–8. When using 7-bit characters with parity, include the parity bit in the character value.
0x52	0–2	—	All ones.
	3–7	—	Reserved
	8–15	RCCM	Received control character mask. Masks comparison of CHARACTER*n*. Each bit of RCCM masks the corresponding bit of CHARACTER*n*. 0 Mask this bit in the comparison of the incoming character and CHARACTER*n*. 1 The address comparison on this bit proceeds normally and no masking occurs. If RCCM is not set, erratic operation can occur during control character recognition.

26.7 BISYNC SYNC Register (BSYNC)

The BSYNC register defines BISYNC stripping and SYNC character insertion. When an underrun occurs, the BISYNC controller inserts SYNC characters until the next buffer is available for transmission. If the receiver is not in hunt mode when a SYNC character is received, it discards this character if the valid bit (BSYNC[V]) is set.When using 7-bit characters with parity, the parity bit should be included in the SYNC register value.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	V	0	0	0	0	0	0	0	SYNC							
Reset	Undefined															
R/W	R/W															
Address	SCC Base + 0x3E															

Figure 26-3. BISYNC SYNC (BSYNC)

Table 26-5 describes BSYNC fields.

Table 26-5. BSYNC Field Descriptions

Bits	Name	Description
0	V	Valid. If V = 1 and the receiver is not in hunt mode when a SYNC character is received, this character is discarded.
1–7	—	All zeros
8–15	SYNC	SYNC character

26.8 SCC BISYNC DLE Register (BDLE)

The BDLE register is used to define the BISYNC stripping and insertion of DLE characters. When an underrun occurs while a message is being sent in transparent mode, the BISYNC controller inserts DLE-SYNC pairs until the next buffer is available for transmission.

In transparent mode, the receiver discards any DLE character received and excludes it from the BCS if the valid bit (BDLE[V]) is set. If the second character is SYNC, the controller discards it and excludes it from the BCS. If it is a DLE, the controller writes it to the buffer and includes it in the BCS. If it is not a DLE or SYNC, the controller examines the control character table and acts accordingly. If the character is not in the table, the buffer is closed with the DLE follow character error bit set. If the valid bit is not set, the receiver treats the character as a normal character. When using 7-bit characters with parity, the parity bit should be included in the DLE register value.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	V	0	0	0	0	0	0	0	DLE							
Reset	Undefined															
R/W	R/W															
Address	SCC Base + 0x40															

Figure 26-4. BISYNC DLE (BDLE)

Table 26-6 describes BDLE fields.

Table 26-6. BDLE Field Descriptions

Bits	Name	Description
0	V	Valid. If V = 1 and the receiver is not in hunt mode when a SYNC character is received, this character is discarded.
1–7	—	All zeros
8–15	DLE	DLE character

26.9 Sending and Receiving the Synchronization Sequence

The BISYNC channel can be programmed to send and receive a synchronization pattern defined in the DSR. GSMR_H[SYNL] defines pattern length, as shown in Table 26-7. The receiver synchronizes on this pattern. Unless SYNL is zero (external sync), the transmitter always sends the entire DSR contents, lsb first, before each frame—the chosen 4- or 8-bit pattern can be repeated in the lower-order bits.

Table 26-7. Receiver SYNC Pattern Lengths of the DSR

GSMR_H[SYNL] Setting	Bit Assignments															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00	An external SYNC signal is used instead of the SYNC pattern in the DSR.															
01	4-Bit															
10	8-Bit															
11	16-Bit															

26.10 Handling Errors in the SCC BISYNC

The controller reports message transmit and receive errors using the channel BDs, error counters, and the SCCE. Modem lines can be directly monitored via the port C pins. Table 26-8 describes transmit errors.

Table 26-8. Transmit Errors

Error	Description
Transmitter Underrun	The channel stops sending the buffer, closes it, sets TxBD[UN], and generates a TXE interrupt if it is enabled. The channel resumes transmission after a RESTART TRANSMIT command is received. Underrun cannot occur between frames or during a DLE–XXX pair in transparent mode.
$\overline{\text{CTS}}$ Lost during Message Transmission	The channel stops sending the buffer, closes it, sets TxBD[CT], and generates a TXE interrupt if not masked. Transmission resumes when a RESTART TRANSMIT command is received.

Table 26-9 describes receive errors.

Table 26-9. Receive Errors

Error	Description
Overrun	The controller maintains a receiver FIFO for receiving data. The CP begins programming the SDMA channel (if the buffer is in external memory) and updating the CRC when the first byte is received in the Rx FIFO. If an Rx FIFO overrun occurs, the controller writes the received byte over the previously received byte. The previous character and its status bits are lost. The channel then closes the buffer, sets RxBD[OV], and generates the RXB interrupt if it is enabled. Finally, the receiver enters hunt mode.
$\overline{\text{CD}}$ Lost during Message Reception	The channel stops receiving, closes the buffer, sets RxBD[CD], and generates the RXB interrupt if not masked. This error has the highest priority. If the rest of the message is lost, no other errors are checked in the message. The receiver immediately enters hunt mode.
Parity	The channel writes the received character to the buffer and sets RxBD[PR]. The channel stops receiving, closes the buffer, sets RxBD[PR], and generates the RXB interrupt if it is enabled. The channel also increments PAREC and the receiver immediately enters hunt mode.
CRC	The channel updates the CR bit in the BD every time a character is received with a byte delay of eight serial clocks between the status update and the CRC calculation. When control character recognition is used to detect the end of the block and cause CRC checking, the channel closes the buffer, sets the CR bit in the BD, and generates the RXB interrupt if it is enabled.

26.11 BISYNC Mode Register (PSMR)

The PSMR is used as the BISYNC mode register, shown in Figure 26-5. PSMR[RBCS, RTR, RPM, TPM] can be modified on-the-fly.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	NOS				CRC		RBCS	RTR	RVD	DRT	—		RPM		TPM	
Reset	0															
R/W	R/W															
Addr	0xA08 (PSMR1)															

Figure 26-5. Protocol-Specific Mode Register for BISYNC (PSMR)

Table 26-10 describes PSMR fields.

Table 26-10. PSMR Field Descriptions

Bits	Name	Description
0–3	NOS	Minimum number of SYN1–SYN2 pairs (defined in DSR) sent between or before messages.If NOS = 0000, one pair is sent. If NOS = 1111, 16 pairs are sent. The entire pair is always sent, regardless of how GSMR[SYNL) is set. NOS can be modified on-the-fly.
4–5	CRC	CRC selection. x0 Reserved. 01 CRC16 (BISYNC). X16 + X15 + X2 + 1. PRCRC and PTCRC should be initialized to all zeros or all ones before the channel is enabled. In either case, the transmitter sends the calculated CRC noninverted and the receiver checks the CRC against zero. Eight-bit data characters (without parity) are configured when CRC16 is chosen. 11 LRC (sum check). (BISYNC). For even LRC, initialize PRCRC and PTCRC to zeros before the channel is enabled; for odd LRC, they should be initialized to ones. Note that the receiver checks character parity when BCS is programmed to LRC and the receiver is not in transparent mode. The transmitter sends character parity when BCS is programmed to LRC and the transmitter is not in transparent mode. Use of parity in BISYNC assumes that 7-bit data characters are being used.
6	RBCS	Receive BCS. The receiver internally stores two BCS calculations separated by an eight serial clock delay to allow examination of a received byte to determine whether it should used in BCS calculation. 0 Disable receive BCS. 1 Enable receive BCS. Should be set (or reset) within the time taken to receive the following data byte. When RBCS is reset, BCS calculations exclude the latest fully received data byte. When RBCS is set, BCS calculations continue as normal.
7	RTR	Receiver transparent mode. 0 Normal receiver mode with SYNC stripping and control character recognition. 1 Transparent receiver mode. SYNCs, DLEs, and control characters are recognized only after a leading DLE character. The receiver calculates the CRC16 sequence even if it is programmed to LRC while in transparent mode. Initialize PRCRC to the CRC16 preset value before setting RTR.
8	RVD	Reverse data. 0 Normal operation. 1 Any portion of the SCC defined to operate in BISYNC mode operates by reversing the character bit order and sending the msb first.
9	DRT	Disable receiver while sending. DRT should not be set for typical BISYNC operation. 0 Normal operation. 1 As the SCC sends data, the receiver is disabled and gated by the internal $\overline{\text{RTS}}$ signal. This helps if the BISYNC channel is being configured onto a multidrop line and the user does not want to receive its own transmission. Although BISYNC usually uses a half-duplex protocol, the receiver is not actually disabled during transmission.
10–11	—	Reserved, should be cleared.

Table 26-10. PSMR Field Descriptions (continued)

Bits	Name	Description
12–13	RPM	Receiver parity mode. Selects the type of parity check that the receiver performs. RPM can be modified on-the-fly and is ignored unless CRC = 11 (LRC). Receive parity errors cannot be disabled but can be ignored. 00 Odd parity. The transmitter counts ones in the data word. If the sum is not odd, the parity bit is set to ensure an odd number. An even sum indicates a transmission error. 01 Low parity. If the parity bit is not low, a parity error is reported. 10 Even parity. An even number must result from the calculation performed at both ends of the line. 11 High parity. If the parity bit is not high, a parity error is reported.
14–15	TPM	Transmitter parity mode. Selects the type of parity the transmitter performs and can be modified on-the-fly. TPM is ignored unless CRC = 11 (LRC). 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 one in the parity bit position).

26.12 SCC BISYNC Receive BD (RxBD)

The CP uses BDs to report on each buffer received. It closes the buffer, generates a maskable interrupt, and starts receiving data into the next buffer after any of the following:

- A user-defined control character is received.
- An error is detected.
- A full receive buffer is detected.
- The ENTER HUNT MODE command is issued.
- The CLOSE RX BD command is issued.

Figure 26-6 shows the SCC BISYNC RxBD.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**E**	—	**W**	**I**	L	F	**CM**	—	DE	—		NO	PR	CR	OV	CD
Offset + 2	Data Length															
Offset + 4	**Rx Data Buffer Pointer**															
Offset + 6																

Figure 26-6. SCC BISYNC RxBD

Table 26-11 describes SCC BISYNC RxBD status and control fields.

Table 26-11. SCC BISYNC RxBD Status and Control Field Descriptions

Bits	Name	Description
0	**E**	Empty. 0 The buffer is full or stopped receiving because of an error. The core can read or write any fields of this RxBD. The CP does not use this BD as long as the E bit is zero. 1 The buffer is not full. The CP controls this BD and buffer. The core should not update this BD.
1	—	Reserved, should be cleared.
2	**W**	Wrap (last BD in table). 0 Not the last BD in the table. 1 Last BD in the table. After this buffer is used, the CP receives incoming data into the first BD that RBASE points to. The number of BDs in this table is determined by the W bit.
3	**I**	Interrupt. 0 No interrupt is generated after this buffer is used. 1 SCCE[RXB] is set when the controller closes this buffer, which can cause an interrupt if it is enabled.
4	L	Last in frame. Set when this buffer is the last in a frame. If $\overline{CD}$ is negated in envelope mode or an error is received, one or more of the OV, CD, and DE bits are set. The controller writes the number of frame octets to the data length field. 0 Not the first buffer in the frame. 1 The first buffer in the frame.
5	F	First in frame. Set when this is the first buffer in a frame. 0 Not the first buffer in a frame. 1 First buffer in a frame
6	**CM**	Continuous mode. 0 Normal operation. 1 The CP does not clear E after this BD is closed; the buffer is overwritten when the CP accesses this BD next. However, E is cleared if an error occurs during reception, regardless of how CM is set.
7	—	Reserved, should be cleared.
8	DE	DPLL error. Set when a DPLL error occurs during reception. In decoding modes where a transition is should occur every bit, the DPLL error is set when a transition is missing.
9–10	—	Reserved, should be cleared.
11	NO	Rx non-octet-aligned frame. Set when a frame is received containing a number of bits not evenly divisible by eight.
12	PR	Parity error. Set when a character with parity error is received. Upon a parity error, the buffer is closed; thus, the corrupted character is the last byte of the buffer. A new Rx buffer receives subsequent data.
13	CR	Rx CRC error. Set when this frame contains a CRC error. Received CRC bytes are always written to the receive buffer.
14	OV	Overrun. Set when a receiver overrun occurs during frame reception.
15	CD	Carrier detect lost. Indicates when the carrier detect signal, $\overline{CD}$, is negated during frame reception.

Data length and buffer pointer fields are described in Section 21.3, “SCC Buffer Descriptors (BDs).” Data length represents the number of octets the CP writes into this buffer, including the BCS. For BISYNC mode, clear these bits. It is incremented each time a received character is written to the buffer.

26.13 SCC BISYNC Transmit BD (TxBD)

The CP arranges data to be sent on the SCC channel in buffers referenced by the channel TxBD table. The CP uses BDs to confirm transmission or indicate errors so the core knows buffers have been serviced. The user configures status and control bits before transmission, but the CP sets them after the buffer is sent.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**R**	—	**W**	**I**	**L**	**TB**	**CM**	**BR**	**TD**	**TR**	**B**	—			UN	CT
Offset + 2	**Data Length**															
Offset + 4	**Tx Data Buffer Pointer**															
Offset + 6																

Figure 26-7. SCC BISYNC TxBD

Table 26-12 describes SCC BISYNC TxBD status and control fields.

Table 26-12. SCC BISYNC TxBD Status and Control Field Descriptions

Bits	Name	Description
0	**R**	Ready. 0 The buffer is not ready for transmission. The current BD and buffer can be updated. The CP clears R after the buffer is sent or after an error condition. 1 The user-prepared buffer has not been sent or is being sent. This BD cannot be updated while R = 1.
1	—	Reserved, should be cleared.
2	**W**	Wrap (last BD in table). 0 Not the last BD in the table. 1 Last BD in the table. After this buffer is used, the CP sends data using the first BD that TBASE points to. The number of TxBDs in this table is determined only by the W bit.
3	**I**	Interrupt. 0 No interrupt is generated after this buffer is serviced. 1 SCCE[TXB] or SCCE[TXE] is set after the CP services this buffer, which can cause an interrupt.
4	**L**	Last in message. 0 The last character in the buffer is not the last character in the current block. 1 The last character in the buffer is the last character in the current block. The transmitter enters and stays in normal mode after sending the last character in the buffer and the BCS, if enabled.
5	**TB**	Transmit BCS. Valid only when the L bit is set. 0 Send an SYN1–SYN2 or idle sequence (specified in GSMR_H[RTSM]) after the last character in the buffer. 1 Send the BCS sequence after the last character. The controller also resets the BCS generator after sending the BCS.
6	**CM**	Continuous mode. 0 Normal operation. 1 The CP does not clear R after this BD is closed, so the buffer is resent when the CP next accesses this BD. However, R is cleared if an error occurs during transmission, regardless of how CM is set.
7	**BR**	BCS reset. Determines whether transmitter BCS accumulation is reset before sending the data buffer. 0 BCS accumulation is not reset. 1 BCS accumulation is reset before sending the data buffer.

Table 26-12. SCC BISYNC TxBD Status and Control Field Descriptions

Bits	Name	Description
8	**TD**	Transmit DLE. 0 No automatic DLE transmission can occur before the data buffer. 1 The transmitter sends a DLE character before sending the buffer, which saves writing the first DLE to a separate buffer in transparent mode. See TR for information on control characters.
9	**TR**	Transparent mode. 0 The transmitter enters and stays in normal mode after sending the buffer. The transmitter automatically inserts SYNCs if an underrun condition occurs. 1 The transmitter enters or stays in transparent mode after sending the buffer. It automatically inserts DLE–SYNC pairs if an underrun occurs (the controller finishes a buffer with L = 0 and the next BD is not available). It also checks all characters before sending them. If a DLE is detected, another DLE is sent automatically. Insert a DLE or program the controller to insert one before each control character. The transmitter calculates the CRC16 BCS even if PSMR[BCS] is programmed to LRC. Initialize PTCRC to CRC16 before setting TR.
10	**B**	BCS enable. 0 The buffer consists of characters that are excluded from BCS accumulation. 1 The buffer consists of characters that are included in BCS accumulation.
11–13	—	Reserved, should be cleared.
14	UN	Underrun. Set when the BISYNC controller encounters a transmitter underrun error while sending the associated data buffer. The CPM writes UN after it sends the associated buffer.
15	CT	$\overline{\text{CTS}}$ lost. The CP sets CT when $\overline{\text{CTS}}$ is lost during message transmission after it sends the data buffer.

Data length and buffer pointer fields are described in Section 21.3, “SCC Buffer Descriptors (BDs).” Although it is never modified by the CP, data length should be greater than zero. The CPM writes these fields after it finishes sending the buffer.

26.14 BISYNC Event Register (SCCE)/BISYNC Mask Register (SCCM)

The BISYNC controller uses the SCC event register (SCCE) to report events recognized by the BISYNC channel and to generate interrupts. When an event is recognized, the controller sets the corresponding SCCE bit. Interrupts are enabled by setting, and masked by clearing, the equivalent bits in the BISYNC mask register (SCCM). SCCE bits are reset by writing ones; writing zeros has no effect. Unmasked bits must be reset before the CP negates the internal interrupt request signal.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—			GLR	GLT	DCC	—		GRA	—		TXE	RCH	BSY	TXB	RXB
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	0xA10 (SCCE1)/0xA14 (SCCM1)															

Figure 26-8. BISYNC Event Register (SCCE)/BISYNC Mask Register (SCCM)

Table 26-13 describes SCCE and SCCM fields.

Table 26-13. SCCE/SCCM Field Descriptions

Bits	Name	Description
0–2	—	Reserved, should be cleared.
3	GLR	Glitch on Rx. Set when the SCC finds an Rx clock glitch.
4	GLT	Glitch on Tx. Set when the SCC finds a Tx clock glitch.
5	DCC	DPLL CS changed. Set when carrier sense status generated by the DPLL changes. Real-time status can be found in SCCS. This is not the $\overline{CD}$ status discussed elsewhere. Valid only when DPLL is used.
6–7	—	Reserved, should be cleared.
8	GRA	Graceful stop complete. Set as soon the transmitter finishes any message in progress when a GRACEFUL STOP TRANSMIT is issued (immediately if no message is in progress).
9–10	—	Reserved, should be cleared.
11	TXE	Tx Error. Set when an error occurs on the transmitter channel.
12	RCH	Receive character. Set when a character is received and written to the buffer.
13	BSY	Busy. Set when a character is received and discarded due to a lack of buffers. The receiver resumes reception after an ENTER HUNT MODE command.
14	TXB	Tx buffer. Set when a buffer is sent. TXB is set as the last bit of data or the BCS begins transmission.
15	RXB	Rx buffer. Set when the CPM closes the receive buffer on the BISYNC channel.

26.15 SCC Status Registers (SCCS)

The SCC status (SCCS) register allows real-time monitoring of RXD. The real-time status of $\overline{CTS}$ and $\overline{CD}$ are part of the port C parallel I/O.

Bit	0	1	2	3	4	5	6	7
Field	—						CS	—
Reset	0000_0000							
R/W	R							
Addr	0xA17 (SCCS1)							

Figure 26-9. SCC Status Registers (SCCS)

Table 26-14 describes SCCS fields.

Table 26-14. SCCS Field Descriptions

Bit	Name	Description
0–5	—	Reserved, should be cleared.
6	CS	Carrier sense (DPLL). 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.
7	—	Reserved, should be cleared.

26.16 Programming the SCC BISYNC Controller

Software has two ways to handle data received by the BISYNC controller. The simplest is to allocate single-byte receive buffers, request an interrupt on reception of each buffer, and implement BISYNC protocol entirely in software on a byte-by-byte basis. This flexible approach can be adapted to any BISYNC implementation. The obvious penalty is the overhead caused by interrupts on each received character.

A more efficient method is to prepare and link multi-byte buffers in the RxBD table and use software to analyze the first two to three bytes of the buffer to determine the type of block received. When this is determined, reception continues without further software intervention until it encounters a control character, which signifies the end of the block and causes software to revert to byte-by-byte reception.

To accomplish this, set SCCM[RCH] to enable an interrupt on every received byte so software can analyze each byte. After analyzing the initial characters of a block, either set PSMR[RTR] or issue a RESET BCS CALCULATION command. For example, if a DLE-STX is received, enter transparent mode. By setting the appropriate PSMR bit, the controller strips the leading DLE from DLE-character sequences. Thus, control characters are recognized only when they follow a DLE character. PSMR[RTR] should be cleared after a DLE-ETX is received.

Alternatively, after an SOH is received, a RESET BCS CALCULATION should be issued to exclude SOH from BCS accumulation and reset the BCS. Notice that PSMR[RBCS] is not needed because the controller automatically excludes SYNCs and leading DLEs.

After the type of block is recognized, SCCE[RCH] should be masked. The core does not interrupt data reception until the end of the current block, which is indicated by the reception of a control character matching the one in the receive control character table. Using Table 26-15, the control character table should be set to recognize the end of the block.

Table 26-15. Control Characters

Control Characters	E	B	H
ETX	0	1	1
ITB	0	1	0
ETB	0	1	1
ENQ	0	0	0
Next entry	0	X	X

After ETX, a BCS is expected; then the buffer should be closed. Hunt mode should be entered when a line turnaround occurs. ENQ characters are used to stop sending a block and to designate the end of the block for a receiver, but no CRC is expected. After control character reception, set SCCM[RCH] to reenable interrupts for each byte of data received.

26.17 SCC BISYNC Programming Example

This BISYNC controller initialization example for SCC1 uses an external clock. The controller is configured with $\overline{\text{RTS1}}$, $\overline{\text{CTS1}}$, and $\overline{\text{CD1}}$ active. Both the receiver and transmitter use CLK3.

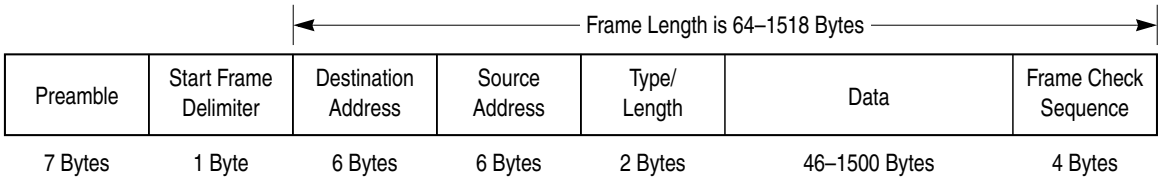
1. Configure the port A pins to enable TXD1 and RXD1. Write PAPAR[14,15] and PAODR[14,15] with ones and PADIR[14,15] with zeros.
2. Configure the port C pins to enable $\overline{\text{RTS1}}$, $\overline{\text{CTS1}}$, and $\overline{\text{CD1}}$. Set PCSO[10,11] and PCPAR[15]; clear PCPAR[10,11] and PCDIR[10,11,15].
3. Configure port A to enable CLK3. Set PAPAR[5] and clear PADIR[5].
4. Connect CLK3 to SCC1 using the serial interface. Set SICR[R1CS, T1CS] to 0b110.
5. Connect the SCC1 to the NMSI and clear SICR[SC1].
6. Initialize the SDMA configuration register (SDCR).
7. Assuming one RxBD at the beginning of dual-port RAM followed by one TxBD, write RBASE with 0x0000 and TBASE with 0x0008.
8. Write 0x0001 to CPCR to execute the INIT RX AND TX PARAMS command for SCC1. This command updates RBPTR and TBPTR of the serial channel with the new values of RBASE and TBASE.
9. Write RFCR and TFCR with 0x10 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 PRCRC with 0x0000 to comply with CRC16.
12. Write PTCRC with 0x0000 to comply with CRC16.
13. Clear PAREC for clarity.
14. Write BSYNC with 0x8033, assuming a SYNC value of 0x33.
15. Write DSR with 0x3333.
16. Write BDLE with 0x8055, assuming a DLE value of 0x55.
17. Write CHARACTER1–8 with 0x8000. They are not used.
18. Write RCCM with 0xE0FF. It is not used.
19. Initialize the RxBD and assume the data buffer is at 0x00001000 in main memory. Then write 0xB000 to RxBD[Status and Control], 0x0000 to RxBD[Data Length] (optional), and 0x00001000 to RxBD[Buffer Pointer].
20. Initialize the TxBD and assume the Tx data buffer is at 0x00002000 in main memory and contains five 8-bit characters. Then write 0xBD20 to TxBD[Status and Control] 0x0005 to TxBD[Data Length], and 0x00002000 to TxBD[Buffer Pointer].

21. Write 0xFFFF to SCCE to clear any previous events.
22. Write 0x0013 to SCCM to enable the TXE, TXB, and RXB interrupts.
23. Write 0x4000_0000 to the CPM interrupt mask register (CIMR) to allow SCC1 to generate a system interrupt. The CICR should also be initialized.
24. Write 0x00000020 to GSMR_H1 to configure a small receive FIFO width.
25. Write 0x00000008 to GSMR_L1 to configure $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ to automatically control transmission and reception (DIAG bits) and the BISYNC mode. Notice that the transmitter (ENT) and receiver (ENR) are not yet enabled.
26. Set PSMR1 to 0x0600 to configure CRC16, CRC checking on receive, and normal operation (not transparent).
27. Write 0x00000038 to GSMR_L1 to enable the transmitter and receiver. This additional write ensures that ENT and ENR are enabled last.

Note that after 5 bytes are sent, the TxBD is closed. The buffer is closed after 16 bytes are received. Any received data beyond 16 bytes causes a busy (out-of-buffers) condition since only one RxBD is prepared.

Chapter 27
SCC Ethernet Mode

The ethernet IEEE 802.3 protocol is a widely used LAN protocol based on the carrier sense multiple access/collision detect (CSMA/CD) approach. Because ethernet and IEEE 802.3 protocols are similar and can coexist on the same LAN, both are referred to as ethernet in this manual, unless otherwise noted. Figure 27-1 shows ethernet and IEEE 802.3 frame structure.



NOTE: The lsb of each octet is transmitted first.

Figure 27-1. Ethernet Frame Structure

The frame begins with a 7-byte preamble of alternating ones and zeros. Because the frame is Manchester encoded, the preamble gives receiving stations a known pattern on which to lock. The start frame delimiter follows the preamble, signifying the beginning of the frame. 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, but this addressing has never been widely used and is not supported.

The next field is the ethernet type field/IEEE 802.3 length field. 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 different from any type fields used in ethernet. This limits the length of the data portion of the frame to 1,500 bytes and total frame length to 1,518 bytes. The last 4 bytes of the frame are the frame check sequence (FCS), a standard 32-bit CCITT-CRC polynomial used in many protocols.

When a station needs to transmit, it checks for LAN activity. When the LAN is silent for a specified period, the station starts sending. At that time, the station continually checks for collisions on the LAN; if one is found, the station forces a jam pattern (all ones) on its frame and stops sending. Most collisions occur close to the beginning of a frame. The station waits a random period of time, called a backoff, before trying to retransmit. Once the backoff time

expires, the station waits for silence on the LAN before retransmitting, which is called a retry. If the frame cannot be sent within 15 retries, an error occurs.

10-Mbps ethernet transmits at 0.8 µs per byte. The preamble plus start frame delimiter is sent in 6.4 µs. The minimum 10-Mbps ethernet interframe gap is 9.6 µs and the slot time is 52 µs.

27.1 Ethernet on the MPC855T

Setting GSMR[MODE] to 0b1100 selects ethernet. The SCC performs the full set of IEEE 802.3/ethernet CSMA/CD media access control and channel interface functions. Figure 27-2 shows the ethernet block diagram.

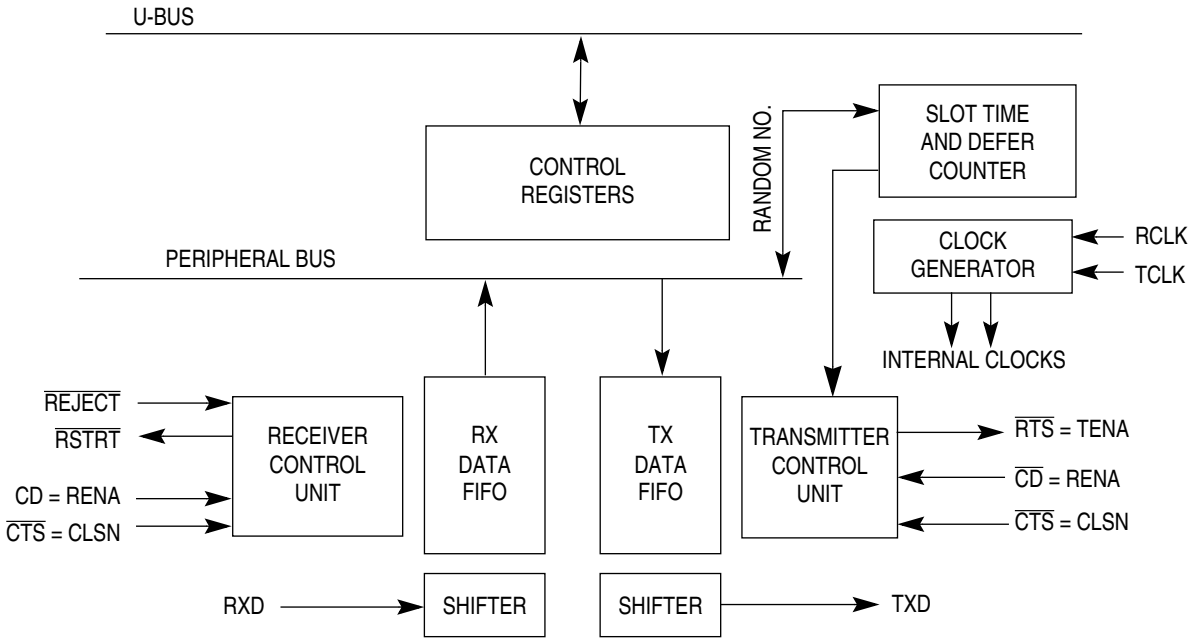


Figure 27-2. Ethernet Block Diagram

The MPC855T ethernet controller 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 MPC855T and 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 MPC855T, Manchester encoding and decoding, automatic selection of 10BASE-T versus AUI ports, 10BASE-T polarity detection and correction. The MC68160 documentation gives more information.

Although the MPC855T contains DPLLs that allow Manchester encoding and decoding, these DPLLs were not designed for ethernet rates. Therefore, the MPC855T ethernet controller bypasses the on-chip DPLLs and uses the external system interface adaptor on

the EEST instead. The on-chip DPLL cannot be used for low-speed (1-Mbps) ethernet either because it cannot properly detect start-of-frame or end-of-frame.

Note that the CPM of the MPC855T requires a minimum system clock frequency of 24 MHz to support ethernet.

27.2 Features

The following list summarizes the main features of the SCC in ethernet mode:

- Performs MAC layer functions of ethernet and IEEE 802.3
- Performs framing functions
  - Preamble generation and stripping
  - Destination address checking
  - CRC generation and checking
  - Automatically pads short frames on transmit
  - Framing error (dribbling bits) handling
- Full collision support
  - Enforces the collision (jamming)
  - Truncated binary exponential backoff algorithm for random wait
  - Two nonaggressive backoff modes
  - Automatic frame retransmission (until the retry limit is reached)
  - Automatic discard of incoming collided frames
  - Delay transmission of new frames for specified interframe gap
- Maximum 10 Mbps bit rate
- Optional full-duplex support
- Back-to-back frame reception
- Detection of receive frames that are too long
- Multibuffer data structure
- Supports 48-bit addresses in three modes
  - Physical–One 48-bit address recognized or 64-bin hash table for physical addresses
  - Logical–64-bin group address hash table plus broadcast address checking
  - Promiscuous–Receives all addresses, but discards frame if $\overline{\text{REJECT}}$ is asserted
- External content-addressable memory (CAM) support on both serial and system bus interfaces
- Up to eight parallel I/O pins can be sampled and appended to any frame
- Optional heartbeat indication
- Transmitter network management and diagnostics

  - Lost carrier sense
  - Underrun
  - Number of collisions exceeded the maximum allowed
  - Number of retries per frame
  - Deferred frame indication
  - Late collision
- Receiver network management and diagnostics
  - CRC error indication
  - Nonoctet alignment error
  - Frame too short
  - Frame too long
  - Overrun
  - Busy (out of buffers)
- Error counters
  - Discarded frames (out of buffers or overrun occurred)
  - CRC errors
  - Alignment errors
- Internal and external loopback mode

27.3 Learning Ethernet on the MPC855T

The standard SCC functionality has been enhanced to support ethernet. First-time MPC855T users who plan to use ethernet should first read the following:

- Chapter 21, “Serial Communications Controller,” describes basic operation of the SCC.
- Chapter 17, “Communications Processor Module and CPM Timers,” describes how the CPM issues special commands to the ethernet channel. The dual-port RAM loads ethernet parameters and initializes BDs for the ethernet channel to use.
- Chapter 19, “SDMA Channels and IDMA Emulation,” discusses how SDMA channels are used to transfer data between the ethernet channel and system memory.
- Section 20.3, “NMSI Configuration,” explains how clocks are routed to the SCC through the bank of clocks.
- Chapter 27, “SCC Ethernet Mode,” should be read next.
- Chapter 33, “Parallel I/O Ports,” shows how to configure the preferred ethernet pin functions to be active.
- Chapter 34, “CPM Interrupt Controller,” defines SCC interrupt priorities and how interrupts are generated to the core.

27.4 Connecting the MPC855T to Ethernet

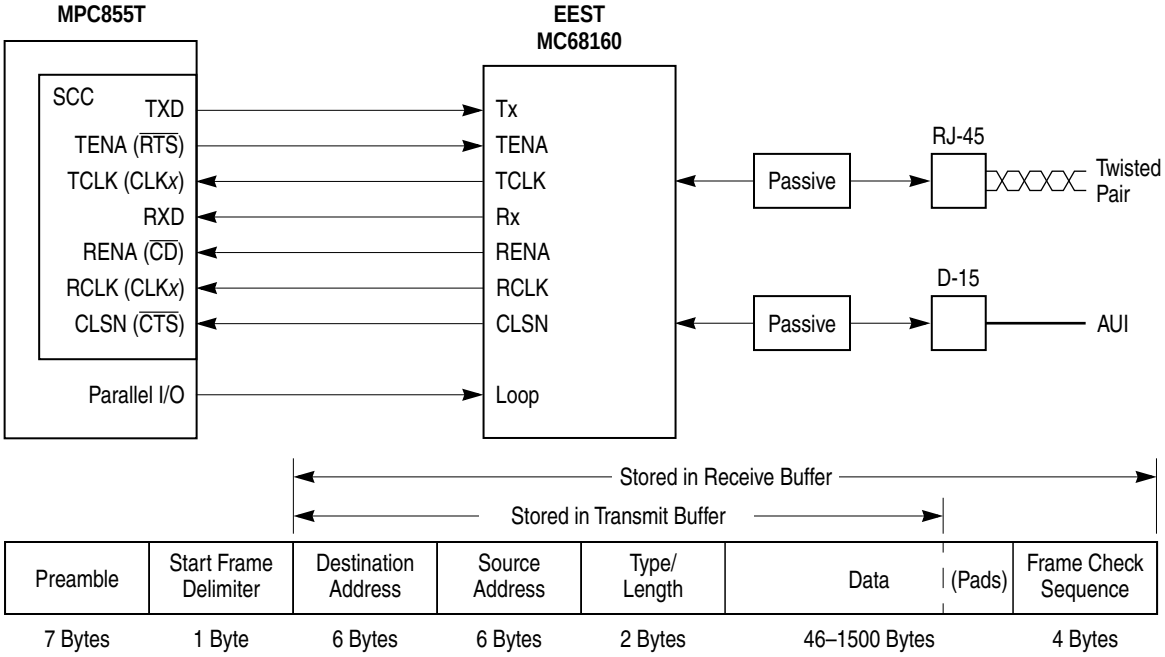
The basic interface to the external EEST chip consists of the following ethernet signals:

- Receive clock (RCLK)—a CLK*x* signal routed through the bank of clocks on the MPC855T.
- Transmit clock (TCLK)—a CLK*x* signal routed through the bank of clocks on the MPC855T. Note that RCLK and TCLK should not be connected to the same CLK*x* since the EEST provides separate transmit and receive clock signals.
- Transmit data (TXD)—the MPC855T TXD signal.
- Receive data (RXD)—the MPC855T RXD signal.

The following signals take on different functionality when the SCC is in ethernet mode:

- Transmit enable (TENA)—$\overline{\text{RTS}}$ becomes TENA. The polarity of TENA is active high, whereas the polarity of $\overline{\text{RTS}}$ is active low.
- Receive enable (RENA)—$\overline{\text{CD}}$ becomes RENA.
- Collision (CLSN)—$\overline{\text{CTS}}$ becomes CLSN. The carrier sense signal is referenced in ethernet descriptions because it indicates when the LAN is in use. Carrier sense is defined as the logical OR of RENA and CLSN.

Figure 27-3 shows the basic components and signals required to make an ethernet connection between the MPC855T and EEST.



NOTE: Short Tx frames are padded automatically.

Figure 27-3. Connecting the MPC855T to Ethernet

The EEST has similar names for its connection to the above seven MPC855T signals. The EEST also provides a loopback input so the MPC855T can perform external loopback testing, which can be controlled by any available MPC855T parallel I/O signal. There are additional pins for interfacing with an optional external content-addressable memory (CAM) described in Section 27.7, “Content-Addressable Memory (CAM) Interface.” The passive components needed to connect to AUI or twisted-pair media are external to the EEST. The MC68160 documentation describes EEST connection circuits.

The MPC855T uses SDMA channels to store bytes received after the start frame delimiter in system memory. When sending, provide the destination address, source address, type/length field, and the transmit data. To meet minimum frame requirements, the MPC855T pads frames with fewer than 46 bytes in the data field and appends the FCS to the frame.

27.5 SCC Ethernet Channel Frame Transmission

The ethernet transmitter works with almost no core intervention. When the core enables the transmitter, the SCC polls the first TxBD in the table every 128 serial clocks. Setting TODR[TOD] lets the next frame be sent without waiting for the next poll.

To begin transmission, the SCC in ethernet mode (called the ethernet controller) fetches data from the buffer, asserts TENA to the EEST, and starts sending the preamble sequence, the start frame delimiter, and frame information. If the line is busy, it waits for carrier sense to remain inactive for 6.0 µs, at which point it waits an additional 3.6 µs before it starts sending (9.6 µs after carrier sense originally became inactive).

If a collision occurs during frame transmission, the ethernet controller follows a specified backoff procedure and tries to retransmit the frame until the retry limit threshold is reached. The ethernet controller stores the first 5 to 8 bytes of the transmit frame in internal RAM so they need not be retrieved from system memory in case of a collision. This improves bus usage and latency when the backoff timer output requires an immediate retransmission. If a collision occurs during frame transmission, the controller returns to the first buffer for a retransmission. The only restriction is that the first buffer must contain at least 9 bytes.

Note that if an ethernet frame consists of multiple buffers, do not reuse the first BD until the CPM clears the R bit of the last BD.

When the end of the current BD is reached and TxBD[L] is set, the FCS bytes are appended (if the TC bit is set in the TxBD), and TENA is negated. This notifies the EEST of the need to generate the illegal Manchester encoding that marks the end of an ethernet frame. After CRC transmission, the ethernet controller writes the frame status bits into the BD and clears the R bit. When the end of the current BD is reached and the L bit is not set, only the R bit is cleared.

In either mode, whether an interrupt is issued depends on how the I bit is set in the TxBD. The ethernet controller proceeds to the next TxBD. Transmission can be interrupted after each frame, after each buffer, or after a specific buffer is sent. The ethernet controller can pad characters to short frames. If TxBD[PAD] is set, the frame is padded up to the value of the minimum frame length register (MINFLR).

To send expedited data before previously linked buffers or for error situations, the GRACEFUL STOP TRANSMIT command can be used to rearrange transmit queue before the CPM sends all the frames; the ethernet controller stops immediately if no transmission is in progress or it will keep sending until the current frame either finishes or terminates with a collision. When the ethernet controller receives a RESTART TRANSMIT command, it resumes transmission. The ethernet controller sends bytes least-significant bit first.

27.6 SCC Ethernet Channel Frame Reception

The ethernet receiver handles address recognition and performs CRC, short frame, maximum DMA transfer, and maximum frame length checking with almost no core intervention. When the core enables the ethernet receiver, it enters hunt mode as soon as RENA is asserted while CLSN is negated. In hunt mode, as data is shifted into the receive shift register one bit at a time, the register contents are compared to the contents of the SYN1 field in the data synchronization register (DSR). This compare function becomes valid a certain number of clocks after the start of the frame (depending on PSMR[NIB]). 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 first received preamble bit, the frame is rejected. If a double-zero fault is detected after 21 bits from the first received preamble bit and before detection of the start frame delimiter (SFD), the frame is also rejected. When the incoming pattern is not rejected and matches the DSR, the SFD has been detected; hunt mode is terminated and character assembly begins.

When the receiver detects the first bytes of the frame, the ethernet controller performs address recognition 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 process completes, which improves bus usage with frames not addressed to this station. The receiver also operates with an external CAM. With an external CAM, frame reception continues normally, unless the CAM specifically signals the frame to be rejected. See Section 27.7, “Content-Addressable Memory (CAM) Interface.”

If a match is found, the ethernet controller fetches the next RxBD and, if it is empty, starts transferring the incoming frame to the RxBD associated data buffer. If a collision is detected during the frame, the RxBDs 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 RxBD 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 RxBD in the table and, if

it is empty, continues transferring the rest of the frame to this buffer. The RxBD length is determined by MRBLR in the SCC general-purpose parameter RAM, which should 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 buffer. The data length written to the last BD in the ethernet frame is the length of the entire frame and it enables the software to correctly recognize the frame-too-long condition.

When the receive frame is complete, the ethernet controller can sample one byte from the port B parallel I/O and append this byte to the end of the last RxBD in the frame. For any PB[16–23] pins defined as outputs, the contents of the PBDAT latch is read instead of the pin. This capability is useful for CAM applications and can be used when the external CAM is not present. Sampling occurs at the end of frame reception.

The ethernet controller then sets the L bit in the RxBD, writes the other frame status bits into the RxBD, 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.

27.7 Content-Addressable Memory (CAM) Interface

The ethernet controller can connect to an external CAM through the serial interface or a system bus interface. Both interfaces can be used at the same time because there is no mode bit to select them, but they are described separately here for clarity. To implement an option, enable the pins needed for the implementation. Both interfaces use the MPC855T $\overline{\text{REJECT}}$ signal to signify that the current frame should be discarded. Internal address recognition logic and an external CAM can be used simultaneously; see Section 27.11, “SCC Ethernet Address Recognition.”

Note that the IDMA and the ethernet CAM interfaces both use the $\overline{\text{SDACK}}$(1–2) signals. If both functions are needed, the function code registers (FCRs) of the IDMA and the ethernet SCC can be programmed with unique values for the address type, AT[1–3]. External logic can then distinguish between the two interfaces.

27.7.1 Serial CAM Interface

When the start frame delimiter is recognized, the MPC855T asserts $\overline{\text{RSTRT}}$ for one bit time on the second destination address bit. The CAM control logic uses $\overline{\text{RSTRT}}$ (in combination with RXD and RCLK) to store the destination or source address and to generate writes to the CAM for address recognition. In addition, RENA supplied from the EEST can abort the comparison if a collision occurs on the receive frame.

After the comparison, if CAM control logic asserts $\overline{\text{REJECT}}$ for the current receive frame, the ethernet controller immediately stops writing data to system memory and reuses the

buffer(s) for the next frame. If the CAM accepts the frame, CAM control logic does nothing and $\overline{\text{REJECT}}$ is not asserted. $\overline{\text{REJECT}}$ must be asserted before the end of the receive frame.

The CAM control logic can provide additional information on PB(16–23). The ethernet controller writes this additional byte to memory during the last SDMA write if PSMR[SIP] is set. The ethernet controller samples this information tag as the last FCS byte is read from the receive FIFO. The CAM control logic must provide the information tag no later than when RENA is negated at the end of a noncollision frame and should be held stable on PB(16–23) until $\overline{\text{SDACK}}$(1–2) indicate that the tag byte is being written to memory.

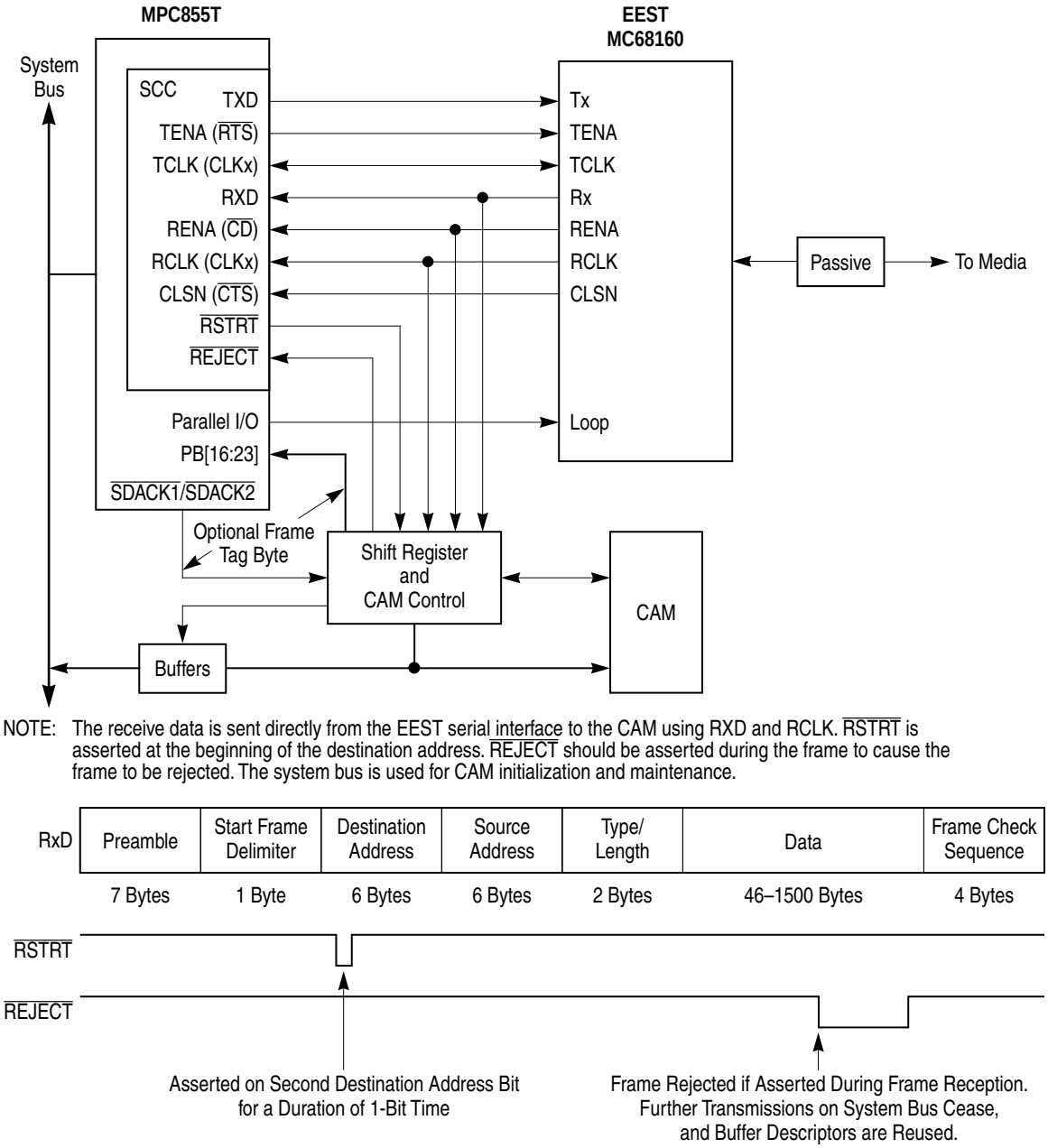
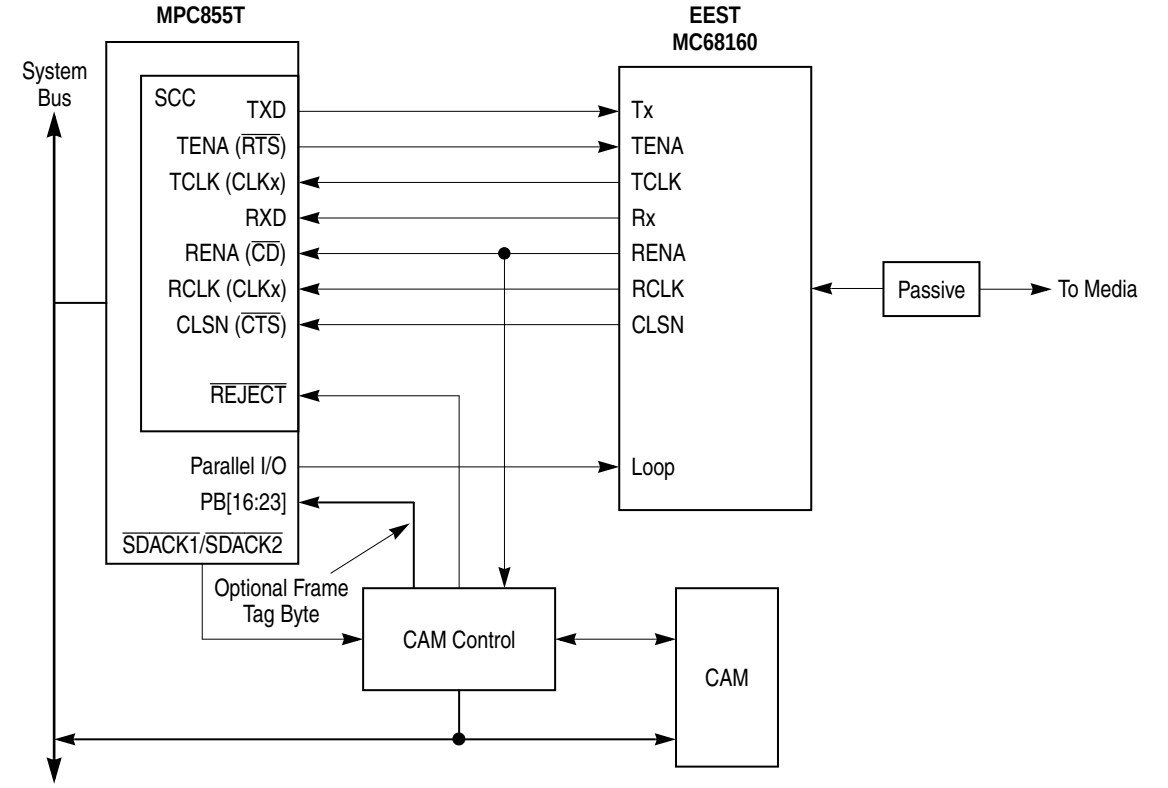


Figure 27-4. MPC855T Ethernet Serial CAM Interface

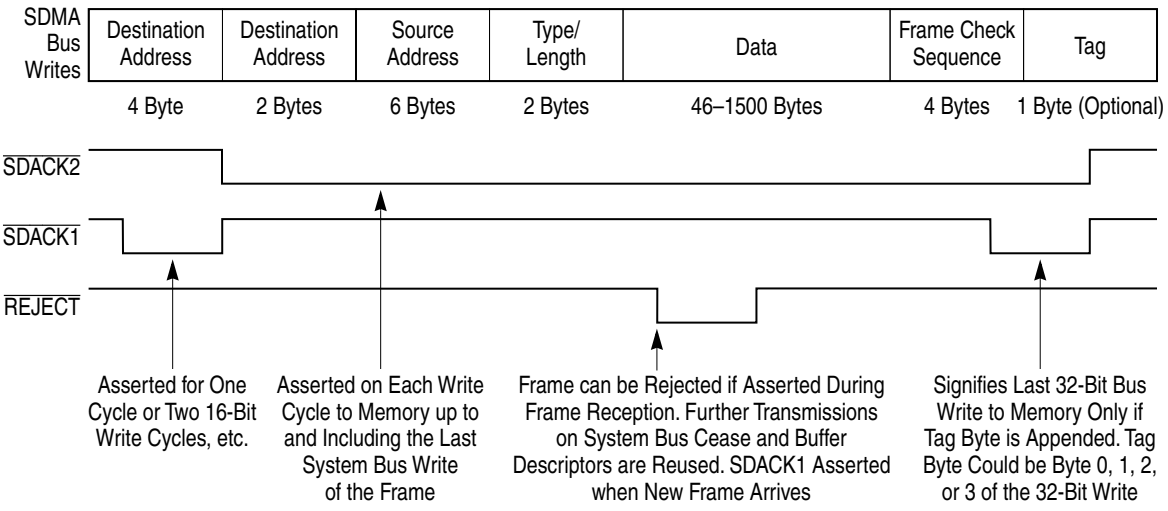
27.7.2 Parallel CAM Interface

The MPC855T outputs SDACK1 and SDACK2 whenever it writes ethernet frame data to system memory. They are asserted during all bus cycles on which ethernet frame data is written to memory and are not used for other protocols. The CAM control logic uses these signals simultaneously to enable the CAM writes with system memory writes. The advantage of the CAM capturing frame data as it is written to system memory is that the

data is already in parallel form when it leaves the MPC855T. Figure 27-5 shows a parallel interface configuration.



NOTE: The receive data is sent to the CAM as it is written to system memory. The SDACK1/SDACK2 signals are used to identify the destination address and any other preferred frame bytes. The RSTRT signal is not required in this configuration, although it is still available.



NOTE: The diagram shows SDMA system bus writes, not data on the RXD pin. Other bus activity can occur between successive 32-bit writes. In such a case, SDACK1/SDACK2 would not be asserted for other bus activity.

Figure 27-5. MPC855T Ethernet Parallel CAM Interface

The $\overline{\text{SDACK1}}$ and $\overline{\text{SDACK2}}$ signals are asserted during all bus cycle writes of the frame data. One $\overline{\text{SDACK1}}$/$\overline{\text{SDACK2}}$ combination identifies the first 32 bits of the frame, another identifies all mid-frame data, and a third identifies the last 32-bit bus write of the frame only if the tag byte is appended. The tag byte is appended from the sample of PB(16–23) if PSMR[SIP] is set and it is always in byte 3 of the last 32-bit write. The RxBD data length does not include tag byte in the length calculation.

If system memory is 32 bits, the MPC855T 32-bit write takes one bus cycle. If it is 16 or 8 bits, a 32-bit write takes two or four bus cycles. In any case, $\overline{\text{SDACK}}$(1–2) are valid on each bus cycle of a 32-bit write cycle and only during bus cycles associated with the ethernet receiver. As an alternate way to identify accesses from the SCC, a unique address type can be chosen with the SDMA receive channel associated with the ethernet controller.

Note that the tag byte is always written to byte 3 of the last SDMA write to the buffer and is not necessarily appended to the last byte of the frame. The data length field does not include the tag byte. Also, $\overline{\text{SDACK}}$(1–2) equal 0b00 whenever the frame is not a multiple of four regardless of whether the tag byte is appended.

27.8 SCC Ethernet Parameter RAM

For ethernet mode, the protocol-specific area of the SCC parameter RAM is mapped as in Table 27-1.

Table 27-1. SCC Ethernet Parameter RAM Memory Map

Offset [1]	Name	Width	Description
0x30	**C_PRES**	Word	Preset CRC. For the 32-bit CRC-CCITT, initialize to 0xFFFFFFFF.
0x34	**C_MASK**	Word	Constant mask for CRC. For the 32-bit CRC-CCITT, initialized to 0xDEBB20E3.
0x38	**CRCEC**	Word	CRC error, alignment error, and discard frame counters. The CPM maintains these 32-bit (modulo 2^{32}) counters that can be initialized while the channel is disabled. CRCEC is incremented for each received frame with a CRC error, not including frames not addressed to the controller, 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 the controller, 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 DISFC to be incremented.
0x3C	**ALEC**		
0x40	**DISFC**		
0x44	**PADS**	Hword	Short frame PAD character. Write the pad character pattern to be sent when short frame padding is implemented into PADS. The pattern may be of any value, but both the high and low bytes should be the same.
0x46	**RET_LIM**	Hword	Retry limit. Number of retries (typically 15 decimal) that can be made to send a frame. An interrupt can be generated if the limit is reached.
0x48	RET_CNT	Hword	Retry limit counter. Temporary down-counter for counting retries.

Table 27-1. SCC Ethernet Parameter RAM Memory Map (continued)

Offset [1]	Name	Width	Description
0x4A	**MFLR**	Hword	Maximum frame length register (typically 1518 decimal). The ethernet controller checks the length of an incoming ethernet frame against this limit. If it is exceeded, the rest of the frame is discarded and LG is set in the last BD of that frame. The controller reports frame status and length in the last BD. MFLR is defined as all in-frame bytes between the start frame delimiter and the end of the frame.
0x4C	**MINFLR**	Hword	Minimum frame length register. The ethernet controller checks the incoming frame's length against MINFLR (typically 64 decimal). If the received frame is smaller than MINFLR, it is discarded unless PSMR[RSH] is set, in which case, SH is set in the last BD for the frame. For transmitting a frame that is too short, the ethernet controller pads the frame to make it MINFLR bytes long, depending on how PAD is set in the TxBD and on the PAD value in the parameter RAM.
0x4E	**MAXD1**	Hword	Max DMA*n* length. Gives the option to stop system bus writes after a frame exceeds a certain size. However, this value is valid only if an address match is found. The ethernet controller checks the length of an incoming ethernet frame against this user-defined value (usually 1520 decimal). If this limit is exceeded, the rest of the incoming frame is discarded. The ethernet controller waits until the end of the frame or until MFLR bytes are received and reports the frame status and the frame length in the last RxBD. MAXD1 is used when an address matches an individual or group address. MAXD2 is used in promiscuous mode when no address match is detected. In a monitor station, MAXD2 can be much less than MAXD1 to receive entire frames addressed to this station, but only the headers of the other frames are received.
0x50	**MAXD2**	Hword	
0x52	MAXD	Hword	Rx max DMA.
0x54	DMA_CNT	Hword	Rx DMA counter. A temporary down-counter used to track frame length.
0x56	MAX_B	Hword	Maximum BD byte count.
0x58	**GADDR1**	Hword	Group address filter 1–4. Used in the hash table function of the group addressing mode. 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.
0x5A	**GADDR2**		
0x5C	**GADDR3**		
0x5E	**GADDR4**		
0x60	TBUF0_DATA0	Word	Save area 0—current frame.
0x64	TBUF0_DATA1	Word	Save area 1—current frame.
0x68	TBUF0_RBA0	Word	
0x6C	TBUF0_CRC	Word	
0x70	TBUF0_BCNT	Hword	
0x72	**PADDR1_H**	Hword	PADDR1 is the 48-bit individual address of this station. PADDR1_L is the lowest order halfword and PADDR1_H is the highest order halfword.
0x74	**PADDR1_M**		
0x76	**PADDR1_L**		

Table 27-1. SCC Ethernet Parameter RAM Memory Map (continued)

Offset [1]	Name	Width	Description
0x78	**P_PER**	Hword	Persistence. Lets the ethernet controller be less aggressive after a collision. Normally, 0x0000. It can be a value between 1 and 9 (1 is most aggressive). The value is added to the retry count in the backoff algorithm to reduce the chance of transmission on the next time slot. Note: Using P_PER is fully allowed in the ethernet/802.3 specifications. A less aggressive backoff algorithm used by multiple stations on a congested ethernet LAN increases overall throughput by reducing the chance of collision. PSMR[SBT] offers another way to reduce the aggressiveness of the ethernet controller.
0x7A	RFBD_PTR	Hword	Rx first BD pointer.
0x7C	TFBD_PTR	Hword	Tx first BD pointer.
0x7E	TLBD_PTR	Hword	Tx last BD pointer.
0x80	TBUF1_DATA0	Word	Save area 0—next frame.
0x84	TBUF1_DATA1	Word	Save area 1—next frame.
0x88	TBUF1_RBA0	Word	
0x8C	TBUF1_CRC	Word	
0x90	TBUF1_BCNT	Hword	
0x92	TX_LEN	Hword	Tx frame length counter.
0x94	**IADDR1**	Hword	Individual address filter 1–4. Used in the hash table function of the individual addressing mode. Zeros can be written 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.
0x96	**IADDR2**		
0x98	**IADDR3**		
0x9A	**IADDR4**		
0x9C	BOFF_CNT	Hword	Backoff counter.
0x9E	**TADDR_H**	Hword	Allows addition and deletion of addresses from individual and group hash tables. After placing an address in TADDR, issue a SET GROUP ADDRESS command. TADDR_L (temp address low) is the least-significant half word and TADDR_H (temp address high) is the most-significant half word.
0x A0	**TADDR_M**		
0x A2	**TADDR_L**		

[1] From SCC base address. SCC base = IMMR + 0x3C00 (SCC1)

27.9 Programming the Ethernet Controller

The core configures the SCC to operate as an ethernet controller by setting GSMR[MODE] to 0b1100. Receive and transmit errors are reported through RxBD and TxBD. Several GSMR fields must be programmed to special values for ethernet. Set DSR[SYN1] to 0x55 and DSR[SYN2] to 0xD5. The 6 bytes of preamble programmed in the GSMR, in combination with the DSR programming, causes 8 bytes of preamble on transmit (including the 1-byte start delimiter with the value 0xD5).

27.10 SCC Ethernet Commands

Transmit and receive commands are issued to the CP command register (CPCR). Table 27-2 describes transmit commands.

Table 27-2. Transmit Commands

Command	Description
STOP TRANSMIT	When used with the ethernet controller, this command violates a specific behavior of an ethernet/IEEE 802.3 station. It should not be used.
GRACEFUL STOP TRANSMIT	Used to ensure that transmission stops smoothly after the current frame finishes or has a collision. SCCE[GRA] is set once transmission stops, at which point ethernet transmit parameters and their BDs can be updated. TBPTR points to the next TxBD. Transmission begins once the R bit of the next BD is set and a RESTART TRANSMIT command is issued. Note that if GRACEFUL STOP TRANSMIT is issued and the current frame ends in a collision, TBPTR points to the start of the collided frame with the R bit still set in the BD. The frame looks as if it was never sent.
RESTART TRANSMIT	Enables transmission of characters on the transmit channel. The ethernet controller expects it after a GRACEFUL STOP TRANSMIT command is issued or a transmitter error. The ethernet controller resumes transmission from the current TBPTR in the channel TxBD table.
INIT TX PARAMETERS	Initializes transmit parameters in this serial channel parameter RAM to reset state. Issue only when the transmitter is disabled. INIT TX and RX PARAMETERS resets both transmit and receive parameters.

Table 27-3 describes receive commands.

Table 27-3. Receive Commands

Command	Description
ENTER HUNT MODE	After hardware or software is reset and the channel is enabled in GSMR_L, the channel is in receive enable mode and uses the first BD in the table. The receiver then enters hunt mode, waiting for an incoming frame. The ENTER HUNT MODE command is generally used to force the ethernet receiver to stop receiving the current frame and enter hunt mode, in which the ethernet controller continually scans the input data stream 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 buffer is closed and the CRC calculation is reset. The next RxBD is used to receive more frames.
CLOSE RXBD	Should not be used with the ethernet controller.
INIT RX PARAMETERS	Initializes receive parameters in this serial channel parameter RAM to their reset state. Issue it only when the receiver is disabled. INIT TX and RX PARAMETERS resets receive and transmit parameters.
SET GROUP ADDRESS	Used to set one of the 64 bits of the four individual/group address hash filter registers. The 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 CP uses an individual address if the I/G bit in the address stored in TADDR is 0; otherwise, it uses a group address. This command can be executed at any time, regardless of whether the ethernet channel is enabled. 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. Do not simply clear the channel's associated hash table bit because the hash table may have multiple addresses mapped to the same hash table bit.

Note that after a CPM reset via CPCR[RST], the ethernet transmit enable (TENA) signal defaults to its $\overline{\text{RTS}}$, active-low functionality. To prevent false TENA assertions to an external transceiver, configure TENA as an input before issuing a CPM reset. See step 3 in Section 27.22, "SCC Ethernet Programming Example."

27.11 SCC Ethernet Address Recognition

The ethernet controller can filter received frames based on different addressing types—physical (individual), group (multicast), broadcast (all-ones group address), and promiscuous. The difference between an individual address and a group address is determined by the I/G bit in the destination address field. A flowchart for address recognition on received frames is shown in Figure 27-6.

In the physical type of address recognition, the ethernet controller compares the destination address field of the received frame with the user-programmed physical address in PADDR1. Address recognition can be performed on multiple individual addresses using the IADDR1–4 hash table.

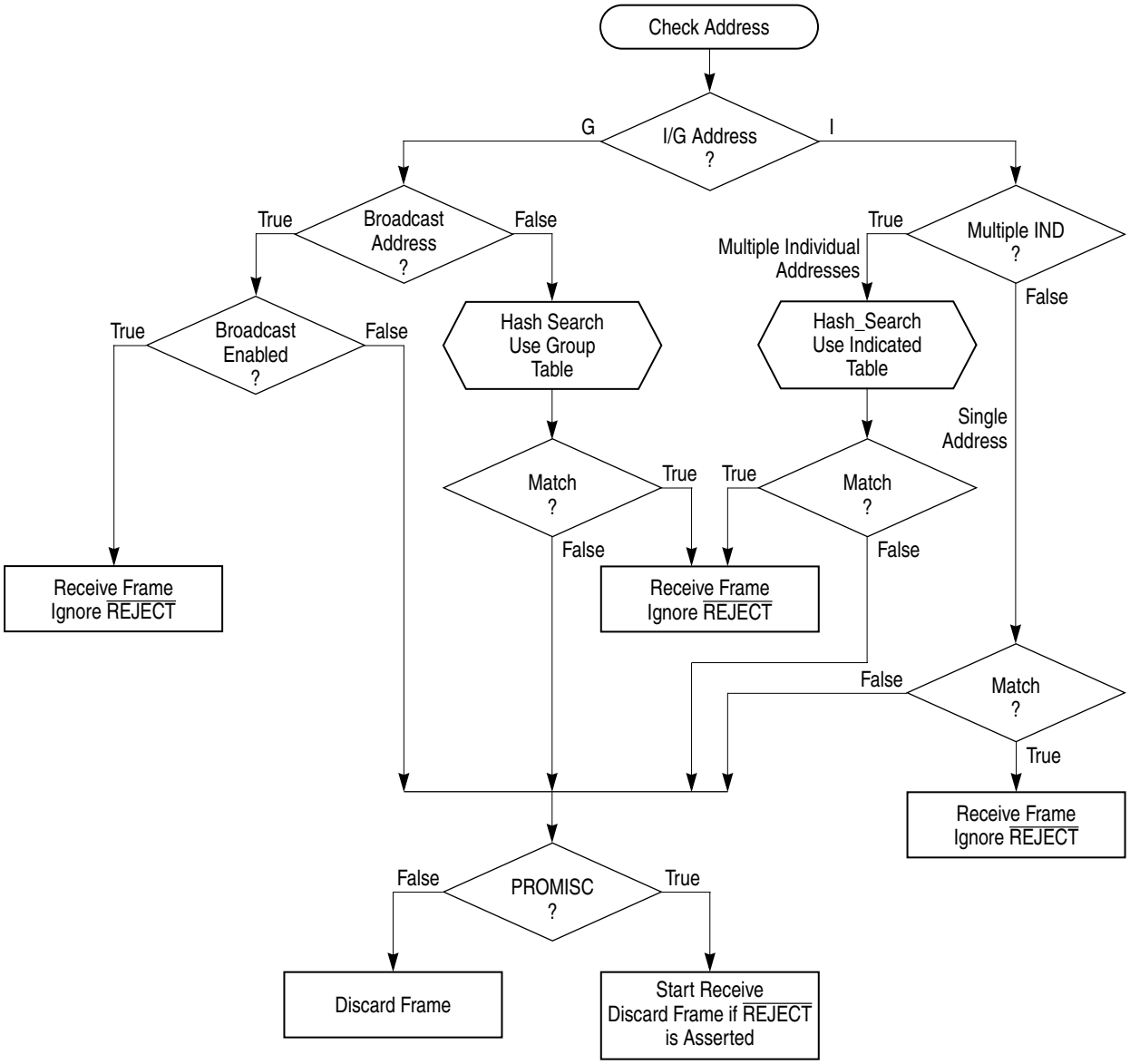


Figure 27-6. Ethernet Address Recognition Flowchart

In group address recognition, the controller determines whether the group address is a broadcast address. If broadcast addresses are enabled, the frame is accepted, but if the group address is not a broadcast address, address recognition can be performed on multiple group addresses using the GADDR*n* hash table. In promiscuous mode, the controller receives all incoming frames regardless of their address, unless $\overline{\text{REJECT}}$ is asserted.

If an external CAM is used for address recognition, select promiscuous mode; the frame can be rejected by asserting $\overline{\text{REJECT}}$ while the frame is being received. The on-chip address recognition functions can be used in addition to the external CAM address recognition functions.

If the external CAM stores addresses that should be rejected rather than accepted, the use of $\overline{\text{REJECT}}$ by the CAM should be logically inverted.

27.12 Hash Table Algorithm

Individual and group hash filtering operate using certain processes. The ethernet controller maps any 48-bit address into one of 64 bins, each represented by a bit stored in GADDR*x* or IADDR*x*. When a SET GROUP ADDRESS command is executed, the 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 GADDRs or IADDRs; bits 29–26 of the CRC result indicate the bit in that register.

When the 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. So, 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. Frames that reach memory must be further filtered by the processor to determine if they 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. In such instances, an external CAM is advised if the extra bus usage cannot be tolerated. See Section 27.7, “Content-Addressable Memory (CAM) Interface.”

Hash tables cannot be used to reject frames that match a set of entered addresses because unintended addresses are mapped to the same bit in the hash table. Thus, an external CAM must be used to implement this function.

27.13 Interpacket Gap Time

The receiver receives back-to-back frames with a minimum interpacket spacing of 9.6 µs. In addition, after the backoff algorithm, the transmitter waits for carrier sense to be negated before resending the frame. Retransmission begins 9.6 µs after carrier sense is negated if it stays negated for at least 6.4 µs.

27.14 Handling Collisions

If a collision occurs as a frame is being sent, the ethernet controller continues sending for at least 32 bit times, thus sending a JAM pattern of 32 ones. If a collision occurs during the preamble sequence, the JAM pattern is sent at the end of the 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 52 µs). If a collision occurs after 64 byte times, 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 reported only in the BD if the length of the frame exceeds MINFLR or if PSMR[RSH] = 1.

27.15 Internal and External Loopback

Both internal and external loopback are supported by the ethernet controller. In loopback mode, both of the SCC FIFOs are used and the channel actually operates in a full-duplex fashion. Both internal and external loopback are configured using combinations of PSMR[LPB] and GSMR[DIAG].

Internal loopback disconnects the SCC from the serial interface. Receive data is connected to the transmit data and the receive clock is connected to the transmit clock. Data from the transmit FIFO is received immediately into the receive FIFO. There is no heartbeat check in this mode; configure TENA as a general-purpose output. That is, set PCDIR[15], and clear PCPAR[15], PCDAT[15], and PSMR[HBC].

In external loopback operation, the ethernet controller listens for data being received from the EEST at the same time that it is sending.

27.16 Full-Duplex Ethernet Support

To run full-duplex ethernet, select loopback and full-duplex ethernet modes in the SCC’s protocol-specific mode register, (PSMR[LPB, FDE] = 1). The loopback mode tells the ethernet controller to accept received frames without signaling a collision. Setting PSMR[FDE] tells the controller that it can send while receiving without waiting for a clear line (carrier sense).

27.17 Handling Errors in the Ethernet Controller

The ethernet controller reports frame reception and transmission error conditions using channel BDs, error counters, and SCCE. Table 27-4 describes transmission errors.

Table 27-4. Transmission Errors

Error	Description
Transmitter underrun	If this error occurs, the channel sends 32 bits that ensures a CRC error, stops sending the buffer, closes it, sets the UN bit in the TxBD and SCCE[TXE]. The channel resumes transmission after it receives a RESTART TRANSMIT command.
Carrier sense lost during frame transmission	When this error occurs and no collision is found in the frame, the channel sets the CSL bit in the TxBD, sets SCCE[TXE], and continues sending the buffer normally. No retries are performed after this error occurs. Carrier sense is the logical OR of RENA and CLSN.
Retransmission retry limit expired	The channel stops sending the buffer, closes it, sets the RL bit in the TxBD and SCCE[TXE]. The channel resumes transmission after it receives a RESTART TRANSMIT command.
Late collision	When this error occurs, the channel stops sending the buffer, closes it, sets SCCE[TXE] and the LC bit in the TxBD. The channel resumes transmission after it receives the RESTART TRANSMIT command. This error is discussed further in the definition of PSMR[LCW].
Heartbeat	Some transceivers have a heartbeat (signal-quality error) self-test. To signify a good self-test, the transceiver indicates a collision to the MPC855T within 20 clocks after the ethernet controller sends a frame. This heartbeat condition does not imply a collision error, but that the transceiver seems to be functioning properly. If SCCE[HBC] = 1 and the MPC855T does not detect a heartbeat condition after sending a frame, a heartbeat error occurs; the channel closes the buffer, sets the HB bit in the TxBD, and generates the TXE interrupt if it is enabled.

Table 27-4 describes reception errors.

Table 27-5. Reception Errors

Error	Description
Overrun	The ethernet controller maintains an internal FIFO for receiving data. When it overruns, the channel writes the received byte over the previously received byte. The previous byte and frame status are lost. The channel closes the buffer, sets RxBD[OV] and SCCE[RXF], and increments the discarded frame counter (DISFC). The receiver then enters hunt mode.
Busy	A frame was received and discarded because of a lack of buffers. The channel sets SCCE[BSY] and increments DISFC. The receiver then enters hunt mode.
Non-Octet Error (Dribbling Bits)	The ethernet controller handles up to seven dribbling bits when the receive frame terminates nonoctet aligned. It checks the CRC of the frame on the last octet boundary. If there is a CRC error, a frame nonoctet aligned error is reported, SCCE[RXF] is set, and the alignment error counter is incremented. If there is no CRC error, no error is reported. The receiver then enters hunt mode.
CRC	When a CRC error occurs, the channel closes the buffer, sets SCCE[RXF] and CR in the RxBD, and 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 CRC errors can be ignored if checking is not required.

27.18 Ethernet Mode Register (PSMR)

In ethernet mode, the protocol-specific mode register (PSMR), shown in Figure 27-7, is used as the ethernet mode register.

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	SIP	LCW	NIB			FDE
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	0xA08 (PSMR1)															

Figure 27-7. Ethernet Mode Register (PSMR)

Table 27-6 describes PSMR fields.

Table 27-6. PSMR Field Descriptions

Bits	Name	Description
0	HBC	Heartbeat checking. 0 No heartbeat checking is performed. Do not wait for a collision after transmission. 1 Wait 20 transmit clocks or 2 µs for a collision asserted by the transceiver after transmission. The HB bit in the TxBD is set if the heartbeat is not heard within 20 transmit clocks.
1	FC	Force collision. 0 Normal operation. 1 The channel forces a collision when each frame is sent. To test collision logic configure the MPC855T in loopback operation. In the end, the retry limit for each transmit frame is exceeded.
2	RSH	Receive short frames. 0 Discard short frames that are not as long as MINFLR. 1 Receive short frames.
3	IAM	Individual address mode. 0 Normal operation. A single 48-bit physical address in PADDR1 is checked when it is received. 1 The individual hash table is used to check all individual addresses that are received.
4–5	CRC	CRC selection. Only CRC = 10 is valid. Complies with ethernet specifications. 32-bit CCITT-CRC. X32 + X26 + X23 + X22 + X16 + X12 + X11 + X10 + X8 + X7 + X5 + X4 + X2 + X1 +1.
6	PRO	Promiscuous. 0 Check the destination address of incoming frames. 1 Receive the frame regardless of its address unless $\overline{\text{REJECT}}$ is asserted as it is being received.
7	BRO	Broadcast address. 0 Receive all frames containing the broadcast address. 1 Reject all frames containing the broadcast address, unless PRO = 1.
8	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. Retransmission is less aggressive than the maximum allowed in IEEE 802.3. The persistence (P_PER) feature in the parameter RAM can be used in combination with or in place of SBT.
9	LPB	Loopback operation. 0 Normal operation. 1 External loopback is used if GSMR[DIAG] is set for normal operation; internal loopback is used if DIAG is configured for loopback operation.
10	SIP	Sample input pins. 0 Normal operation. 1 After a frame is received, the value on PB(16–23) is sampled and written to the end of the last buffer of the frame. This value is called a tag byte. If the frame is discarded, the tag byte is also discarded.

Table 27-6. PSMR Field Descriptions (continued)

Bits	Name	Description
11	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.
12–14	NIB	Number of ignored bits. Determines how soon after RENA assertion the ethernet controller should begin looking for the start frame delimiter. Typically NIB = 101 (22 bits). 000 Begin searching 13 bits after the assertion of RENA. 001 Begin searching 14 bits after the assertion of RENA. ... 111 Begin searching 24 bits after the assertion of RENA.
15	FDE	Full duplex ethernet. 0 Disable full-duplex ethernet mode. 1 Enable full-duplex ethernet mode. Note: When FDE = 1, PSMR[LPB] must be set also.

27.19 SCC Ethernet Receive Buffer Descriptor

The ethernet controller uses the RxBD to report on the received data for each buffer.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**E**	—	**W**	**I**	L	F	—	M	—		LG	NO	SH	CR	OV	CL
Offset + 2	Data Length															
Offset + 4	**Rx Data Buffer Pointer**															
Offset + 6																

Figure 27-8. SCC Ethernet RxBD

Table 27-7 describes RxBD status and control fields.

Table 27-7. SCC Ethernet RxBD Status and Control Field Descriptions

Bits	Name	Description
0	**E**	Empty. 0 The buffer is full or stopped receiving data because an error occurred. The core can read or write any fields of this RxBD. The CPM does not use this BD as long as the E bit is zero. 1 The buffer is not full. The CPM controls this BD and its buffer; do not modify this BD.
1	—	Reserved, should be cleared.
2	**W**	Wrap (final BD in table). 0 Not the last BD in the table. 1 Last BD in the table. After this buffer is used, the CPM receives incoming data into the first BD that RBASE points to. The number of BDs is determined only by the W bit.
3	**I**	Interrupt. Note that this bit does not mask SCCE[RXF] interrupts. 0 No SCCE[RXB] interrupt is generated after this buffer is used. 1 SCCE[RXB] or SCCE[RXF] is set when this buffer is used by the ethernet controller. These two bits can cause interrupts if they are enabled.

Table 27-7. SCC Ethernet RxBD Status and Control Field Descriptions (continued)

Bits	Name	Description
4	L	Last in frame. The ethernet controller sets this bit when this buffer is the last one in a frame, which occurs when the end of a frame is reached or an error is received. In the case of error, one or more of the CL, OV, CR, SH, NO, and LG bits are set. The 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.
5	F	First in frame. The 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.
6	—	Reserved, should be cleared.
7	M	Miss. (valid only if L = 1) The ethernet controller sets M for frames that are accepted in promiscuous mode, but are flagged as a miss by internal address recognition. Thus, in promiscuous mode, M determines whether a frame is destined for this station. 0 The frame is received because of an address recognition hit. 1 The frame is received because of promiscuous mode.
8–9	—	Reserved, should be cleared.
10	LG	Rx frame length violation. Set when a frame length greater than the maximum defined for this channel has been recognized. Only the maximum number of bytes allowed is written to the buffer.
11	NO	Rx nonoctet-aligned frame. Set when a frame containing a number of bits not divisible by eight is received. Also, the CRC check that occurs at the preceding byte boundary generated an error.
12	SH	Short frame. Set if a frame smaller than the minimum defined for this channel was recognized. Occurs if PSMR[RSH] = 1.
13	CR	Rx CRC error. set when a frame contains a CRC error.
14	OV	Overrun. Set when a receiver overrun occurs during frame reception.
15	CL	Collision. This frame is closed because a collision occurred during frame reception. CL is set only if a late collision occurs or if PSMR[RSH] is enabled. Late collisions are better defined in PSMR[LCW].

Data length and buffer pointer fields are described in Section 21.3, “SCC Buffer Descriptors (BDs).” Data length includes the total number of frame octets (including four bytes for CRC).

Figure 27-9 shows an example of how RxBDs are used in receiving.

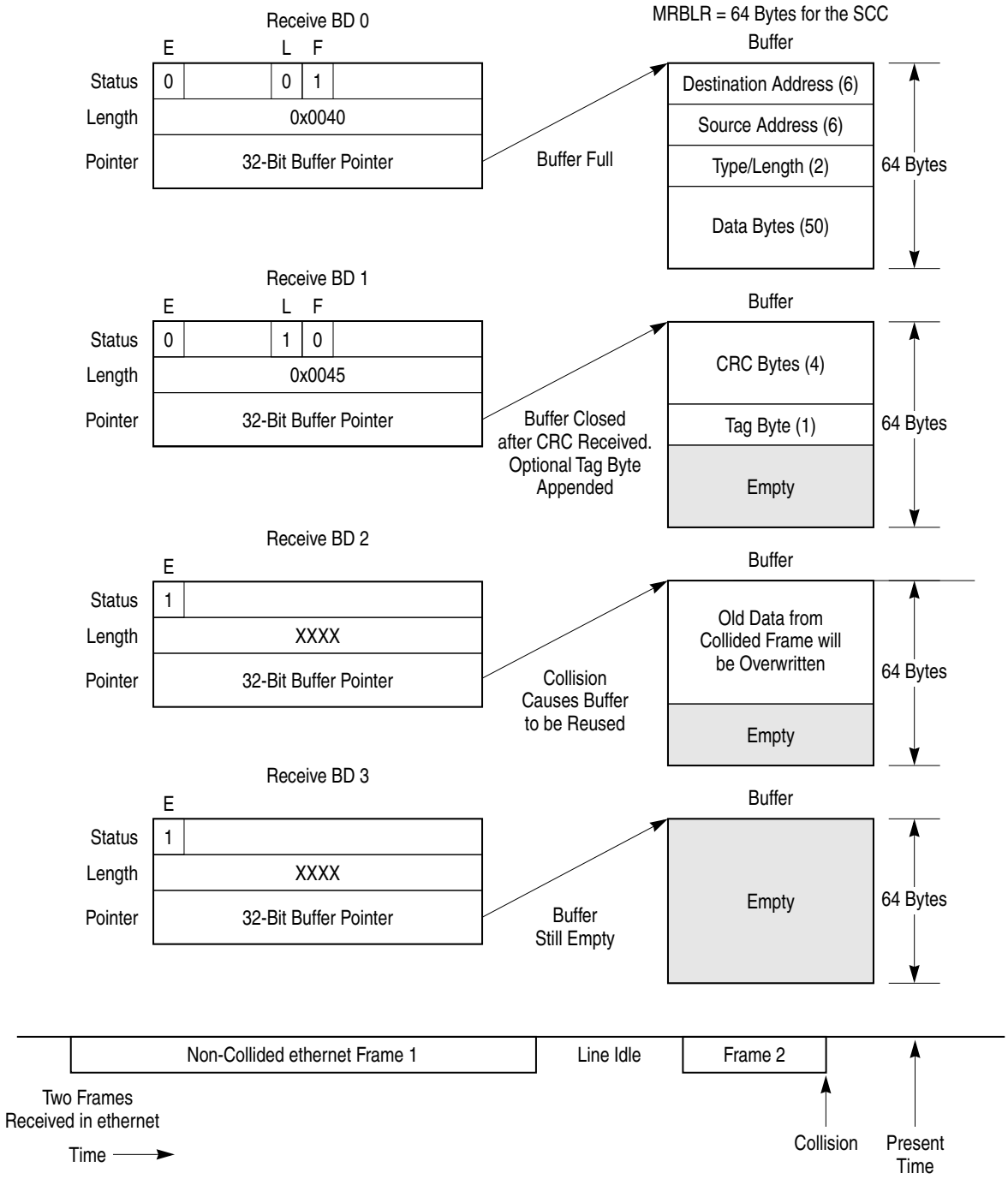


Figure 27-9. Ethernet Receiving using RxBDs

27.20 SCC Ethernet Transmit Buffer Descriptor

Data is sent to the ethernet controller for transmission on the SCC channel by arranging it in buffers referenced by the channel TxBD table. The ethernet controller uses TxBDs to confirm transmission or indicate errors so the core knows 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																

Figure 27-10. SCC Ethernet TxBD

Table 27-8 describes TxBD status and control fields.

Table 27-8. SCC Ethernet TxBD Status and Control Field Descriptions

Bits	Name	Description
0	**R**	Ready. 0 The buffer is not ready for transmission. The user can update this BD or its data buffer. The CPM clears R after the buffer has been sent or after an error occurs. 1 The user-prepared buffer has not been sent or is currently being sent. Do not modify this BD.
1	**PAD**	Short frame padding. Valid only when L 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 sent frame equals the MINFLR and they are stored in PADs in the parameter RAM.
2	**W**	Wrap (final BD in table). 0 Not the last BD in the table. 1 Last BD in the table. After this buffer is used, the CPM receives incoming data into the first BD that TBASE points to in the table. The number of TxBDs in this table is determined only by the W bit. Note: The TxBD table must contain more than one BD in ethernet mode.
3	**I**	Interrupt. 0 No interrupt is generated after this buffer is serviced. 1 SCCE[TXB] or SCCE[TXE] is set after this buffer is serviced. These bits can cause interrupts if they are enabled.
4	**L**	Last. 0 Not the last buffer in the transmit frame. 1 Last buffer in the transmit frame.
5	**TC**	Tx CRC. Valid only when L = 1. Otherwise, it is ignored. 0 End transmission immediately after the last data byte. 1 Transmit the CRC sequence after the last data byte.
6	DEF	Defer indication. The frame was deferred before being sent successfully, that is, the transmitter had to wait for carrier sense before sending because the line was busy. This is not a collision indication; collisions are indicated in RC.
7	HB	Heartbeat. Set when the collision input was not asserted within 20 transmit clocks after transmission. HB cannot be set unless PSMR[HBC] = 1. The SCC writes HB after it finishes sending the buffer.

Table 27-8. SCC Ethernet TxBD Status and Control Field Descriptions (continued)

Bits	Name	Description
8	LC	Late collision. Set when a collision occurred after the number of bytes defined for PSMR[LCW] are sent. The ethernet controller stops sending and writes this bit after it finishes sending the buffer.
9	RL	Retransmission limit. Set when the transmitter fails (Retry Limit + 1) attempts to successfully transmit a message because of repeated collisions on the medium. The ethernet controller writes this bit after it finishes attempting to send the buffer.
10–13	RC	Retry count. Indicates the number of retries required before the frame was sent successfully. If RC = 0, the frame was sent correctly the first time. If RC = 15 and RET_LIM = 15 in the parameter RAM, 15 retries were required. Because the counter saturates at 15, if RC = 15 and RET_LIM > 15, then 15 or more retries were required. The controller writes this field after it successfully sends the buffer.
14	UN	Underrun. Set when the ethernet controller encounters a transmitter underrun while sending the buffer. The ethernet controller writes UN after it finishes sending the buffer.
15	CSL	Carrier sense lost. Set when carrier sense is lost during frame transmission. The ethernet controller writes CSL after it finishes sending the buffer.

Data length and buffer pointer fields are described in Section 21.3, “SCC Buffer Descriptors (BDs).”

27.21 SCC Ethernet Event Register (SCCE)/Mask Register (SCCM)

The SCC event register (SCCE) is used as the ethernet event register to generate interrupts and report events recognized by the ethernet channel. When an event is recognized, the ethernet controller sets the corresponding SCCE bit. Interrupts are enabled by setting, and masked by clearing, the equivalent bits in the ethernet mask register (SCCM). SCCE bits are cleared by writing ones; writing zeros has no effect. All unmasked bits must be cleared before the CPM clears the internal interrupt request.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—								GRA	—		TXE	RXF	BSY	TXB	RXB
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	0xA10 (SCCE1)/0xA14 (SCCM1)															

Figure 27-11. SCC Ethernet Event Register (SCCE)/Mask Register (SCCM)

Figure 27-9 describes SCCE and SCCM fields.

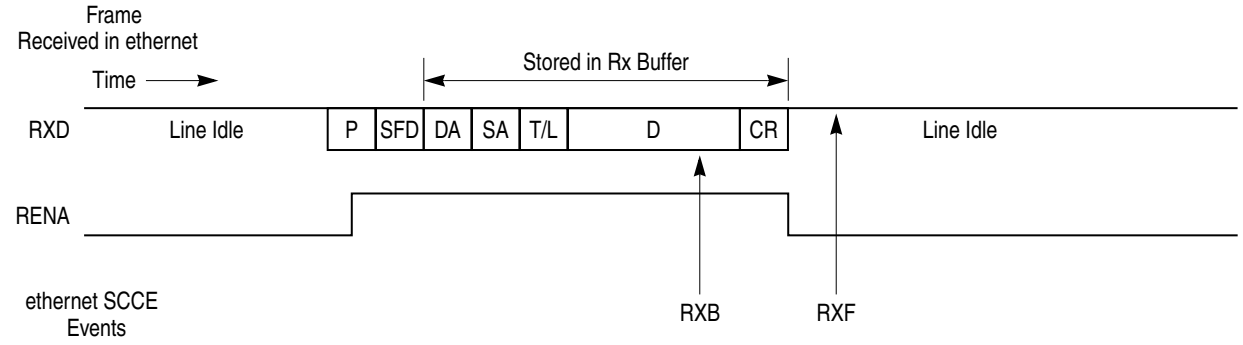
Table 27-9. SCCE/SCCM Field Descriptions

Bits	Name	Description
0–7	—	Reserved, should be cleared.
8	GRA	Graceful stop complete. Set as soon the transmitter finishes any frame that was in progress when a GRACEFUL STOP TRANSMIT command was issued. It is set immediately if no frame was in progress.

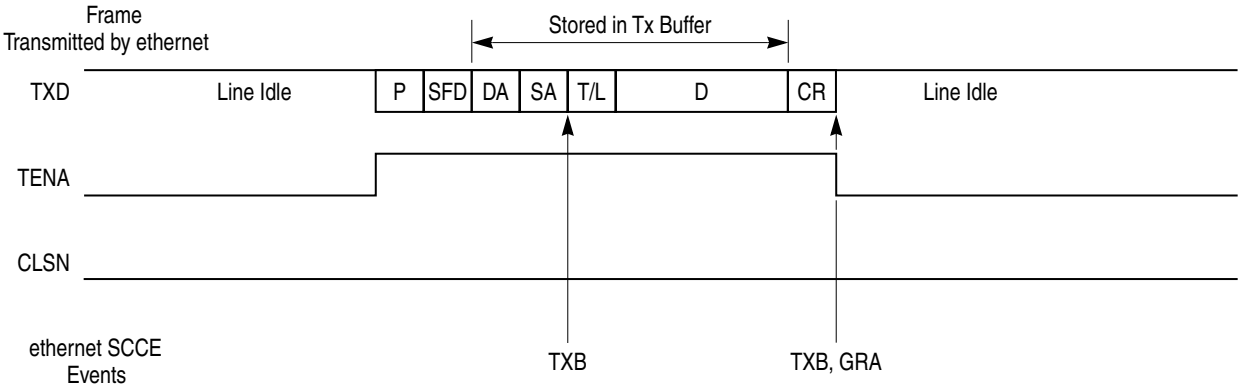
Table 27-9. SCCE/SCCM Field Descriptions (continued)

Bits	Name	Description
9–10	—	Reserved, should be cleared.
11	TXE	Set when an error occurs on the transmitter channel.
12	RXF	Rx frame. Set when a complete frame has been received on the ethernet channel.
13	BSY	Busy condition. Set when a frame is received and discarded due to a lack of buffers.
14	TXB	Tx buffer. Set when a buffer has been sent on the ethernet channel.
15	RXB	Rx buffer. Set when a buffer that was not a complete frame was received on the ethernet channel.

Figure 27-12 shows an example of interrupts that can be generated in ethernet protocol.



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.



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.

LEGEND:
P = Preamble, SFD = Start frame delimiter, DA and SA = Source/Destination address,
T/L = Type/Length, D = Data, CR = CRC bytes

Figure 27-12. Ethernet Interrupt Events Example

Note that the SCC status register (SCCS) cannot be used with the ethernet protocol. The current state of the RENA and CLSN signals can be found in port C.

27.22 SCC Ethernet Programming Example

The following is an initialization sequence for the SCC1 in ethernet mode. The CLK1 pin is used for the ethernet receiver and CLK2 is used for the transmitter.

1. Configure port A to enable TXD1 and RXD1. Set PAPAR[14, 15] and clear PADIR[14, 15] and PAODR[14].
2. Configure port C to enable $\overline{\text{CTS1}}$ (CLSN) and $\overline{\text{CD1}}$ (RENA). Clear PCPAR[10, 11] and PCDIR[10, 11] and set PCSO[10, 11].
3. Do not enable the $\overline{\text{RTS1}}$ (TENA) pin yet because it is still functioning as $\overline{\text{RTS}}$ and transmission on the LAN could begin accidentally.
4. Configure port A to enable the CLK1 and CLK2 pins. Set PAPAR[6, 7] and clear PADIR[6, 7].
5. Connect CLK1 and CLK2 to SCC1 using the serial interface. Set SICR[R1CS] to 0b101 and SICR[T1CS] to 0b100.
6. Connect the SCC1 to the NMSI and clear SICR[SC1].
7. Initialize the SDMA configuration register (SDCR) to 0x0001.
8. Write RBASE and TBASE in the SCC1 parameter RAM to point to the RxBD and TxBD in the dual-port RAM. Assuming one RxBD at the beginning of the dual-port RAM and one TxBD following that RxBD, write RBASE with 0x0000 and TBASE with 0x0008.
9. Program the CPCR to execute an INIT RX AND TX PARAMETERS command for this channel.
10. Write RFCR and TFCR with 0x10 for normal operation.
11. Write MRBLR with the maximum number of bytes per receive buffer. Here, assume 1520 bytes, so MRBLR = 0x05F0. In this example, the user wants to receive an entire frame into one buffer, so MRBLR is the first value larger than 1518 evenly divisible by four.
12. Write C_PRES with 0xFFFF_FFFF to comply with 32-bit CCITT-CRC.
13. Write C_MASK with 0xDEBB_20E3 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 1518 bytes.
18. Write MINFLR with 0x0040 to make the minimum frame size 64 bytes.

19. Write MAXD1 and MAXD2 with 0x05F0 to make the maximum DMA count 1520 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 0x8003_E012_3456.
22. Clear P_PER. 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 RxBD and assume the Rx data buffer is at 0x0000_1000 in main memory. Write 0xB000 to RxBD[Status and Control], 0x0000 to RxBD[Data Length] (optional), and 0x0000_1000 to RxBD[Buffer Pointer].
26. Initialize the TxBD and assume the Tx data frame is at 0x0000_2000 in main memory and contains fourteen 8-bit characters (destination and source addresses plus the type field). Write 0xFC00 to TxBD[Status and Control], add PAD to the frame and generate a CRC. Then write 0x000D to TxBD[Data Length] and 0x0000_2000 to TxBD[Buffer Pointer].
27. Write 0xFFFF to the SCCE register to clear any previous events.
28. Write 0x001A to the SCCM register to enable the TXE, RXF, and TXB interrupts.
29. Write 0x4000_0000 to the CIMR so that SCC1 can generate a system interrupt. The CICR register should also be initialized.
30. Write 0x0000_0000 to GSMR_H1 to enable normal operation of all modes.
31. Write 0x1088_000C to the GSMR_L1 register to configure $\overline{\text{CTS}}$ (CLSN) and $\overline{\text{CD}}$ (RENA) to automatically control transmission and reception (DIAG bits) and the ethernet mode. TCI is set to allow more setup time for the EEST to receive the MPC855T transmit data. TPL and TPP are set for ethernet requirements. The DPLL is not used with ethernet. Note that the ENT and ENR are not enabled yet.
32. Write 0xD555 to the DSR.
33. Set the PSMR1 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 ($\overline{\text{RTS}}$). Since GSMR[MODE] are written to ethernet, the TENA signal is low. Set PCPAR[15] and clear PCDIR[15].
35. Write 0x1088_003C to GSMR_L1 to enable the SCC1 transmitter and receiver. This additional write ensures that ENT and ENR are enabled last.

After 14 bytes and the 46 bytes of automatic pad (plus the 4 bytes of CRC) are sent, the TxBD is closed. Additionally, the receive buffer is closed after a frame is received. Any data received after 1520 bytes or a single frame causes a busy (out-of-buffers) condition because only one RxBD is prepared.

Chapter 28
SCC Transparent Mode

Transparent mode (also called totally transparent or promiscuous mode) provides a clear channel on which the SCC can send or receive serial data without bit-level manipulation. Software implements protocols run over transparent mode. The SCC in transparent mode functions as a high-speed serial-to-parallel and parallel-to-serial converter.

Transparent mode can be used for serially moving data that requires no superimposed protocol, for applications that require serial-to-parallel and parallel-to-serial conversion for communication among chips on the same board, and for applications that require data to be switched without interfering with the protocol encoding itself, such as when data from a high-speed time-multiplexed serial stream is multiplexed into low-speed data streams. The concept is to switch the data path without altering the protocol encoded on that data path.

Transparent mode is configured in the GSMR; see Section 21.2.1, “General SCC Mode Register (GSMR).” Transparent mode is selected in GSMR_H[TTX, TRX] for the transmitter and receiver, respectively. Setting both bits enables full-duplex transparent operation. If only one is set, the other half of the SCC uses the protocol specified in GSMR_L[MODE]. This allows loop-back modes to DMA data from one memory location to another while data is converted to a specific serial format.

The SCC operations are asynchronous with the core. The SCC clock can be supplied from the internal baud rate generator bank, DPLL output, or external pins.

The SCC can work with the time-slot assigner (TSA) or nonmultiplexed serial interface (NMSI) and supports modem lines with the general-purpose I/O pins. Data can be transferred either the msb or lsb first in each octet.

28.1 Features

The following list summarizes the main features of the SCC in transparent mode:

- Flexible buffers
- Automatic SYNC detection on receive
- CRCs can be sent and received
- Reverse data mode
- Another protocol can be performed on the other half of the SCC
- MC68360-compatible SYNC options

28.2 SCC Transparent Channel Frame Transmission Process

The transparent transmitter is designed to work almost no intervention from the core. When the core enables the SCC transmitter in transparent mode, it starts sending idles, which are logic high or encoded ones, as programmed in GSMR_L[TEND]. The SCC polls the first BD in the TxBD table. When there is a message to send, the SCC fetches data from memory, loads the transmit FIFO, and waits for transmitter synchronization, which is achieved with $\overline{\text{CTS}}$ or by waiting for the receiver to achieve synchronization, depending on GSMR_H[TXSY]. Transmission begins when transmitter synchronization is achieved.

When all BD data has been sent, if TxBD[L] is set, the SCC writes the message status bits into the BD, clears TxBD[R], and sends idles until the next BD is ready. If it is ready, some idles are still sent. The transmitter resumes sending only after it achieves synchronization.

If TxBD[L] is cleared when the end of the BD is reached, only TxBD[R] 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 causes a transmit underrun which sets SCCE[TXE].

In both cases, an interrupt is issued according to TxBD[I]. By appropriately setting TxBD[I] in each BD, interrupts are generated after each buffer or group of buffers is sent. The SCC then proceeds to the next BD in the table and any whole number of bytes can be sent. If GSMR_H[REVD] is set, the bit order of each byte is reversed before being sent; the msb of each octet is sent first.

Setting GSMR_H[TFL] makes the transmit FIFO smaller and reduces transmitter latency, but it can cause transmitter underruns at higher transmission speeds. An optional CRC, selected in GSMR_H[TCRC], can be appended to each transparent frame if it is enabled in the TxBD.

When the time-slot assigner (TSA) is used with a transparent-mode channel, synchronization is provided by the TSA. There is a start-up delay for the transmitter, but delays will always be some whole number of complete TSA frames. This means that *n*-byte transmit buffers can be mapped directly into *n*-byte time slots in the TSA frames.

28.3 SCC Transparent Channel Frame Reception Process

When the core enables the SCC 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 full, the SCC clears RxBD[E] and generates a maskable interrupt if RxBD[I] is set. It moves to the next RxBD in the table and begins moving data to its buffer.

If the next buffer is not available, SCCE[BSY] signifies a busy signal that can generate a maskable interrupt. The receiver reverts to hunt mode when an ENTER HUNT MODE command or an error is received. If GSMR_H[REVD] is set, the bit order of each byte is reversed before it is written to memory.

Setting GSMR_H[RFW] reduces receiver latency by making the receive FIFO smaller, which may cause receiver overruns at higher transmission speeds. The receiver always checks the CRC of the received frame, according to GSMR_H[TCRC]. If a CRC is not required, resulting errors can be ignored.

28.4 Achieving Synchronization in Transparent Mode

Once the SCC transmitter is enabled for transparent operation, the TxBD is prepared and the transmit FIFO is preloaded by the SDMA channel, another process must occur before data can be sent. It is called transmit synchronization. Similarly, once the SCC receiver is enabled for transparent operation in the GSMR and the RxBD is made empty for the SCC, receive synchronization must occur before data can be received. An in-line synchronization pattern or an external synchronization signal can provide bit-level control of the synchronization process when sending or receiving.

28.4.1 Synchronization in NMSI Mode

The following sections describe synchronization in NMSI mode.

28.4.1.1 In-Line Synchronization Pattern

The transparent channel can be programmed to receive a synchronization pattern. This pattern is defined in the data synchronization register, DSR; see Section 21.2.3, “Data Synchronization Register (DSR).” Pattern length is specified in GSMR_H[SYNL], as shown in Table 28-1. See also Section 21.2.1, “General SCC Mode Register (GSMR).”

Table 28-1. Receiver SYNC Pattern Lengths of the DSR

GSMR_H[SYNL] Setting	Bit Assignments															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00	An external SYNC signal is used instead of the SYNC pattern in the DSR.															
01	4-bit															
10	8-bit															
11	16-bit															

If a 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. The transmitter synchronizes on the receiver pattern if GSMR_H[RSYN] = 1.

Note that the transparent controller does not automatically send the synchronization pattern; therefore, the synchronization pattern must be included in the transmit buffer.

28.4.1.2 External Synchronization Signals

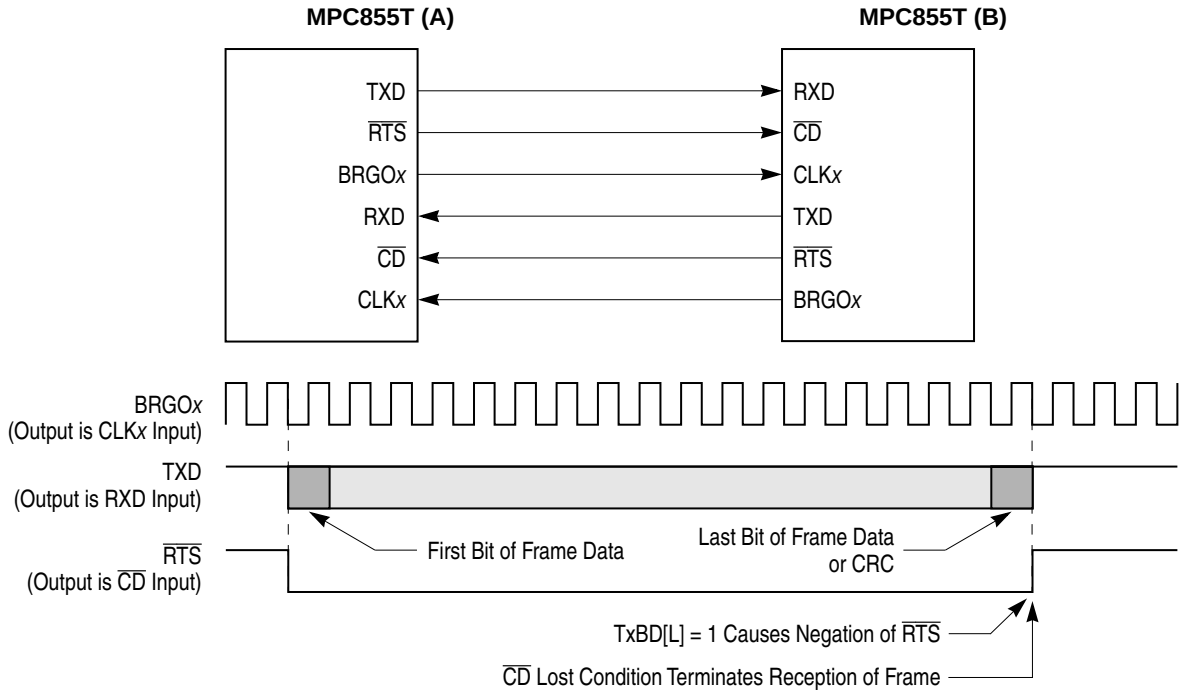
If GSMR_H[SYNL] is 0b00, the transmitter uses $\overline{\text{CTS}}$ and the receiver uses $\overline{\text{CD}}$ to begin the sequence. These signals share two options—pulsing and sampling.

GSMR_H[CDP] and GSMR_H[CTSP] determine whether $\overline{\text{CD}}$ or $\overline{\text{CTS}}$ need to be asserted only once to begin reception/transmission or whether they must remain asserted for the duration of the transparent frame. Pulse operation allows an uninterrupted stream of data. However, use envelope mode to identify frames of transparent data.

The sampling option determines the delay between $\overline{\text{CD}}$ and $\overline{\text{CTS}}$ being asserted and the resulting action by the SCC. Assume either that these signals are asynchronous to the data and internally synchronized by the SCC or that they are synchronous to the data with faster operation. This option allows $\overline{\text{RTS}}$ of one SCC to be connected to $\overline{\text{CD}}$ of another SCC and to have the data synchronized and bit aligned. It is also an option to link the transmitter synchronization to the receiver synchronization. Diagrams for the pulse/envelope and sampling options are shown in Section 21.4.4, “Controlling SCC Timing with RTS, CTS, and CD.”

28.4.1.2.1 External Synchronization Example

Figure 28-1 shows synchronization using external signals.



Notes:
- 1. Each MPC855T generates its own transmit clocks. If the transmit and receive clocks are the same, one MPC855T can generate transmit and receive clocks for the other MPC855T. For example, CLK*x* on MPC855T(B) could be used to clock the transmitter and receiver.
- 2. $\overline{\text{CTS}}$ should be configured as always asserted in the Port C parallel I/O or connected to ground externally.
- 3. The required GSMR 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.
- 4. The transparent frame contains a CRC if TxBD[TC] is set.

Figure 28-1. Sending Transparent Frames between MPC855T

MPC855T(A) and MPC855T(B) exchange transparent frames and synchronize each other using $\overline{\text{RTS}}$ and $\overline{\text{CD}}$. However, $\overline{\text{CTS}}$ is not required because transmission begins at any time. Thus, $\overline{\text{RTS}}$ is connected directly to the other MPC855T $\overline{\text{CD}}$ pin. GSMR_H[RSYN] is not used and transmission and reception from each MPC855T are independent.

28.4.1.3 Transparent Mode without Explicit Synchronization

If there is no need to synchronize the transparent controller at a specific point, the user can ‘fake’ synchronization in one of the following ways:

- Tie a parallel I/O pin to the $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ lines. Then, after enabling the receiver and transmitter, provide a falling edge by manipulating the I/O pin in software.
- Enable the receiver and transmitter for the SCC in loopback mode and then change GSMR_L[DIAG] to 0b00 while the transmitter and receiver and enabled.

28.4.1.4 End of Frame Detection

An end of frame cannot be detected in the transparent data stream since there is no defined closing flag in transparent mode. Therefore, if framing is needed, the user must use the $\overline{CD}$ line to alert the transparent controller of an end of frame.

28.4.2 Synchronization and the TSA

A transparent-mode SCC using the time-slot assigner can synchronize either on a user-defined in-line pattern or by inherent synchronization.

Note that when using the TSA, a newly-enabled transmitter sends from 10 to 15 frames of idles before sending the actual transparent data due to start-up requirements of the TDM. Therefore, when loopback testing through the TDM, expect to receive several bytes of 0xFF before the actual data.

28.4.2.1 In-line Synchronization Pattern

The receiver can be programmed to begin receiving data into the receive buffers only after a specified data pattern arrives. To synchronize on an in-line pattern:

- Set GSMR_H[SYNL].
- Program the DSR with the desired pattern.
- Clear GSMR_H[CDP].
- Set GSMR_H[CTSP, CTSS, CDS].

If GSMR_H[TXSY] is also used, the transmitter begins transmission eight clocks after the receiver achieves synchronization.

28.4.2.2 Inherent Synchronization

Inherent synchronization assumes synchronization by default when the channel is enabled; all data sent from the TDM to the SCC is received. To implement inherent synchronization:

- Set GSMR_H[CDP, CDS, CTSP, CTSS].

If these bits are not set, the received bit stream will be bit-shifted. The SCC loses the first received bit because $\overline{CD}$ and $\overline{CTS}$ are treated as asynchronous signals.

28.5 CRC Calculation in Transparent Mode

The CRC calculations follow the ITU/IEEE standard. The CRC is calculated on the transmitted data stream; that is, from lsb to msb for non-bit-reversed (GSMR_H[REVD] = 0) and from msb to lsb for bit-reversed (GSMR_H[REVD] = 1) transmission. The appended CRC is sent msb to lsb.When receiving, the CRC is calculated as the incoming

bits arrive. The optional reversal of data (GSMR_H[REVD] = 1) is done just before data is stored in memory (after the CRC calculation).

28.6 SCC Transparent Parameter RAM

For transparent mode, the protocol-specific area of the SCC parameter RAM is mapped as in Table 28-2.

Table 28-2. SCC Transparent Parameter RAM Memory Map

Offset [1]	Name	Width	Description
0x 30	CRC_P	Long	CRC preset for totally transparent. For the 16-bit CRC-CCITT, initialize with 0x0000_FFFF. For the 32-bit CRC-CCITT, initialize with 0xFFFF_FFFF and for the CRC-16, initialize with ones (0x0000_FFFF) or zeros (0x0000_0000).
0x 34	CRC_C	Long	CRC constant for totally transparent receiver. For the 16-bit CRC-CCITT, initialize with 0x0000_F0B8. For the 32-bit CRC-CCITT, CRC_C initialize with 0xDEBB_20E3 and for the CRC-16, which is normally used with BISYNC, initialize with 0x0000_0000.

1 From SCC base address. SCC base = IMMR + 0x3C00 (SCC1)

CRC_P and CRC_C overlap with the CRC parameters for the HDLC-based protocols. However, this overlap is not detrimental since the CRC constant is used only for the receiver and the CRC preset is used only for the transmitter, so only one entry is required for each. Thus, the user can choose an HDLC transmitter with a transparent receiver or a transparent transmitter with an HDLC receiver.

28.7 SCC Transparent Commands

The following transmit and receive commands are issued to the CP command register. Table 28-3 describes transmit commands.

Table 28-3. Transmit Commands

Command	Description
STOP TRANSMIT	After hardware or software is reset and the channel is enabled in the GSMR, the channel is in transmit enable mode and starts polling the first BD every 64 clocks (or immediately if TODR[TOD] = 1). STOP TRANSMIT disables frame transmission on the transmit channel. If the transparent controller receives the command during frame transmission, transmission is aborted after a maximum of 64 additional bits and the transmit FIFO is flushed. The current TxBD pointer (TBPTR) is not advanced, no new BD is accessed and no new buffers are sent for this channel. The transmitter will send idles.
GRACEFUL STOP TRANSMIT	Stops 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 no frame is being sent. A transparent frame is not complete until a BD with TxBD[L] set has its buffer completely sent. SCCE[GRA] is set once transmission stops; transmit parameters and their BDs can then be modified. The current TxBD pointer (TBPTR) advances to the next TxBD in the table. Transmission resumes once TxBD[R] is set and a RESTART TRANSMIT command is issued.

Table 28-3. Transmit Commands (continued)

Command	Description
RESTART TRANSMIT	Reenables transmission of characters on the transmit channel. The transparent controller expects it after a STOP TRANSMIT command is issued (at which point the channel is disabled in SCCM), after a GRACEFUL STOP TRANSMIT command is issued, or after a transmitter error. The transparent controller resumes transmission from the current TBPTR in the channel TxBD table.
INIT TX PARAMETERS	Initializes all transmit parameters in the serial channel parameter RAM to reset state. Issue only when the transmitter is disabled. INIT TX AND RX PARAMETERS resets receive and transmit parameters.

Table 28-4 describes receive commands.

Table 28-4. Receive Commands

Command	Description
ENTER HUNT MODE	After hardware or software is reset and the channel is enabled, the channel is in receive enable mode and uses the first BD in the table. ENTER HUNT MODE forces the transparent receiver to the current frame and enter hunt mode where the transparent controller waits for the synchronization sequence. After receiving the command, the current buffer is closed. Further data reception uses the next BD.
CLOSE RXBD	Forces the SCC to close the RxBD if it is being used and to use the next BD for any subsequently received data. If the SCC is not receiving data, no action is taken by this command.
INIT RX PARAMETERS	Initializes all receive parameters in this serial channel parameter RAM to reset state. Issue only when the receiver is disabled. INIT TX AND RX PARAMETERS resets receive and transmit parameters.

28.8 Handling Errors in the Transparent Controller

The SCC reports message reception and transmission errors using the channel buffer descriptors, the error counters, and SCCE. Table 28-5 describes transmit errors.

Table 28-5. Transmit Errors

Error	Description
Transmitter Underrun	When this occurs, the channel stops sending the buffer, closes it, sets TxBD[UN], and generates a TXE interrupt if it is enabled. Transmission resumes after a RESTART TRANSMIT command is received. Underrun occurs after a transmit frame for which TxBD[L] was not set. In this case, only SCCE[TXE] is set. Underrun cannot occur between transparent frames.
$\overline{\text{CTS}}$ Lost During Message Transmission	When this occurs, the channel stops sending the buffer, closes it, sets TxBD[CT], and generates the TXE interrupt if it is enabled. The channel resumes sending after RESTART TRANSMIT is received.

Table 28-6 describes receive errors.

Table 28-6. Receive Errors

Error	Description
Overrun	The SCC maintains a receive FIFO. The CPM starts programming the SDMA channel if the buffer is in external memory and updating the CRC when 8 or 32 bits are received in the FIFO as determined by GSMR_H[RFW]. If a FIFO overrun occurs, the SCC writes the received byte over the previously received byte. The previous character and its status bits are lost. Afterwards, the channel closes the buffer, sets OV in the BD, and generates the RXB interrupt if it is enabled. The receiver immediately enters hunt mode.
$\overline{\text{CD}}$ Lost During Message Reception	When this occurs, the channel stops receiving messages, closes the buffer, sets RxBD[CD], and generates the RXB interrupt if it is enabled. This error has highest priority. The rest of the message is lost, and no other errors are checked in the message. The receiver immediately enters hunt mode.

28.9 Transparent Mode and the PSMR

The protocol-specific mode register (PSMR) is not used by the transparent controller because all transparent mode selections are made in the GSMR. If only half of the SCC (transmitter or receiver) is running the transparent protocol, the other half (receiver or transmitter) can support another protocol. In such a case, use the PSMR for the non-transparent protocol.

28.10 SCC Transparent Receive Buffer Descriptor (RxBD)

The CPM reports information about the received data for each buffer using an RxBD, closes the current buffer, generates a maskable interrupt, and starts receiving data into the next buffer after one of the following occurs:

- An error is detected.
- A full receive buffer is detected.
- An ENTER HUNT MODE command is Issued.
- A CLOSE RXBD command is issued.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**E**	—	**W**	**I**	L	F	**CM**	—	DE	—		NO	—	CR	OV	CD
Offset + 2	Data Length															
Offset + 4	**Rx Buffer Pointer**															
Offset + 6																

Figure 28-2. SCC Transparent Receive Buffer Descriptor (RxBD)

Table 28-7 describes RxBD status and control fields.

Table 28-7. SCC Transparent RxBD Status and Control Field Descriptions

Bits	Name	Description
0	**E**	Empty. 0 The buffer is full or stopped receiving data because an error occurred. The core can read or write to any fields of this RxBD. The CPM does not use this BD when RxBD[E] is zero. 1 The buffer is not full. This RxBD and buffer are owned by the CPM. Once E is set, the core should not write any fields of this RxBD.
1	—	Reserved, should be cleared.
2	**W**	Wrap (final BD in table). 0 Not the last BD in the table. 1 Last BD in the table. After this buffer is used, the CPM receives data into the first BD that RBASE points to. The number of BDs in this table is determined only by RxBD[W].
3	**I**	Interrupt. 0 No interrupt is generated after this buffer is used. 1 When this buffer is closed by the transparent controller, the SCCE[RXB] is set. SCCE[RXB] can cause an interrupt if it is enabled.
4	L	Last in frame. Set by the transparent controller when this buffer is the last in a frame, which occurs when $\overline{CD}$ is negated (if GSMR_H[CDP] = 0) or an error is received. If an error is received, one or more of RxBD[OV, CD, DE] are set. The transparent controller writes the number of frame octets to the BD's data length field. 0 Not the last buffer in a frame. 1 Last buffer in a frame.
5	F	First in frame. The transparent controller sets F when this buffer is the first in the frame: 0 Not the first buffer in a frame. 1 First buffer in a frame.
6	**CM**	Continuous mode. 0 Normal operation. 1 The CPM does not clear RxBD[E] after this BD is closed, letting the buffer be overwritten when the CPM next accesses this BD. However, RxBD[E] is cleared if an error occurs during reception, regardless of how CM is set.
7	—	Reserved, should be cleared.
8	DE	DPLL error. Set by the transparent controller when a DPLL error occurs as this buffer is received. In decoding modes, where a transition is promised every bit, DE is set when a missing transition occurs. If a DPLL error occurs, no other error checking is performed.
9–10	—	Reserved, should be cleared.
11	NO	Rx non-octet. Set when a frame containing a number of bits not exactly divisible by eight is received.
12	—	Reserved, should be cleared.
13	CR	CRC error indication bits. Indicates that this frame contains a CRC error. The received CRC bytes are always written to the receive buffer. CRC checking cannot be disabled, but it can be ignored.
14	OV	Overrun. Indicates that a receiver overrun occurred during buffer reception.
15	CD	Carrier detect lost. Indicates when $\overline{CD}$ is negated during buffer reception.

Data length and buffer pointer fields are described in Section 21.3, "SCC Buffer Descriptors (BDs)." The Rx buffer pointer must be divisible by four, unless

GSMR_H[RFW] is set to 8 bits wide, in which case the pointer can be even or odd. The buffer can reside in internal or external memory.

28.11 SCC Transparent Transmit Buffer Descriptor (TxBD)

Data is sent to the CPM for transmission on the SCC channel by arranging it in buffers referenced by the TxBD table. The CPM uses BDs to confirm transmission or indicate error conditions so the processor knows buffers have been serviced. Prepare status and control bits before transmission; they are set by the CPM after the buffer is sent.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**R**	—	**W**	**I**	**L**	**TC**	**CM**	—							UN	CT
Offset + 2	**Data Length**															
Offset + 4	**Tx Buffer Pointer**															
Offset + 6																

Figure 28-3. SCC Transparent Transmit Buffer Descriptor (TxBD)

Table 28-8 describes SCC Transparent TxBD status and control fields.

Table 28-8. SCC Transparent Tx BD Status and Control Field Descriptions

Bit	Name	Description
0	**R**	Ready. 0 The buffer is not ready for transmission. The BD and buffer can be updated. The CPM clears R after the buffer is sent or after an error is encountered. 1 The user-prepared buffer is not sent yet or is being sent. This BD cannot be updated while R = 1.
1	—	Reserved, should be cleared.
2	**W**	Wrap (final BD in table). 0 Not the last BD in the table. 1 Last BD in the table. After this buffer is used, the CPM receives incoming data into the first BD that TBASE points to. The number of TxBDs in this table is determined only by TxBD[W].
3	**I**	Interrupt. Note that clearing this bit does not disable SCCE[TXE]. 0 No interrupt is generated after this buffer is serviced. 1 When the CPM services this buffer, SCCE[TXB] or SCCE[TXE] is set. These bits can cause interrupts if they are enabled.
4	**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 sent 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 sent, the transmitter requires synchronization before the next buffer is sent.
5	**TC**	Transmit CRC. 0 No CRC sequence is sent after this buffer. 1 A frame check sequence defined by GSMR_H[TCRC] is sent after the last byte of this buffer.

Table 28-8. SCC Transparent Tx BD Status and Control Field Descriptions (continued)

Bit	Name	Description
6	**CM**	Continuous mode. 0 Normal operation. 1 The CPM does not clear TxBD[R] after this BD is closed, so the buffer is automatically resent when the CPM accesses this BD next. However, TxBD[R] is cleared if an error occurs during transmission, regardless of how CM is set.
7–13	—	Reserved, should be cleared.
14	UN	Underrun. Set when the SCC encounters a transmitter underrun condition while sending the buffer.
15	CT	CTS lost. Indicates the $\overline{\text{CTS}}$ was lost during frame transmission.

Data length and buffer pointer fields are described in Section 21.3, “SCC Buffer Descriptors (BDs).” Although it is never modified by the CP, data length should be greater than zero. The buffer pointer can be even or odd and can reside in internal or external memory.

28.12 SCC Transparent Event Register (SCCE)/ Mask Register (SCCM)

When the SCC is in transparent mode, the SCC event register (SCCE) functions as the transparent event register to report events recognized by the transparent channel and to generate interrupts. When an event is recognized, the transparent controller sets the corresponding SCCE bit. Interrupts are enabled by setting, and masked by clearing, the equivalent bits in the transparent mask register (SCCM).

Event bits are reset by writing ones; writing zeros has no effect. All unmasked bits must be reset before the CPM negates the internal interrupt request signal. Figure 28-4 shows the event and mask registers.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—			GLR	GLT	DCC	—		GRA	—		TXE	—	BSY	TXB	RXB
Reset	0000_0000_0000_0000															
R/W	R/W															
Address	0xA10 (SCCE1)/0xA14 (SCCM1)															

Figure 28-4. SCC Transparent Event Register (SCCE)/Mask Register (SCCM)

Table 28-9 describes SCCE/SCCM fields.

Table 28-9. SCCE/SCCM Field Descriptions

Bit	Name	Description
0–2	—	Reserved, should be cleared.
3	GLR	Glitch on Rx. Set when the SCC finds a glitch on the receive clock.
4	GLT	Glitch on Tx. Set when the SCC finds a glitch on the transmit clock.
5	DCC	DPLL CS changed. Set when the DPLL-generated carrier sense status changes (valid only when the DPLL is used). Real-time status can be read in SCCS. This is not the $\overline{CD}$ status mentioned elsewhere.
6–7	—	Reserved, should be cleared.
8	GRA	Graceful stop complete. Set when a graceful stop initiated by completes as soon as the transmitter finishes any frame in progress when the GRACEFUL STOP TRANSMIT command was issued. Immediately if no frame was in progress when the command was issued.
9–10	—	Reserved, should be cleared.
11	TXE	Tx error. Set when an error occurs on the transmitter channel.
12	—	Reserved, should be cleared.
13	BSY	Busy condition. Set when a byte or word is received and discarded due to a lack of buffers. The receiver resumes reception after it gets an ENTER HUNT MODE command.
14	TXB	Tx buffer. Set no sooner than when the last bit of the last byte of the buffer begins transmission, assuming L is set in the TxBD. If it is not, TXB is set when the last byte is written to the transmit FIFO.
15	RXB	Rx buffer. Set when a complete buffer was received on the SCC channel, no sooner than two serial clocks after the last bit of the last byte in which the buffer is received on RXD.

28.13 SCC Status Register in Transparent Mode (SCCS)

The SCC status register (SCCS) allows monitoring of real-time status conditions on the RXD line. The real-time status of $\overline{CTS}$ and $\overline{CD}$ are part of the port C parallel I/O.

Bit	0	1	2	3	4	5	6	7
Field	—						CS	—
Reset	0000_0000							
R/W	R							
Address	0xA17 (SCCS1)							

Figure 28-5. SCC Status Register in Transparent Mode (SCCS)

Table 28-10 describes SCCS fields.

Table 28-10. SCCS Field Descriptions

Bit	Name	Description
0–5	—	Reserved, should be cleared.
6	CS	Carrier sense (DPLL). 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.
7	—	Reserved, should be cleared.

28.14 SCC1 Transparent Programming Example

The following initialization sequence enables the transmitter and receiver, which operate independently of each other. The sequence implements the connection shown for MPC855T(B) in Figure 28-1. The transparent controller is configured with $\overline{\text{RTS1}}$ and $\overline{\text{CD1}}$ active, and $\overline{\text{CTS1}}$ is configured to be grounded internally in port C. CLK3 externally provides the transmit and receive clocks. A 16-bit CRC-CCITT is sent with each transparent frame. The FIFOs are configured for fast operation.

1. Configure port A to enable TXD1 and RXD1. Set PAPAR[14,15] and clear PADIR[14,15] and PAODR[14,15].
2. Configure port C to enable $\overline{\text{RTS1}}$, $\overline{\text{CTS1}}$, and $\overline{\text{CD1}}$. Set PCPAR[15] and PCSO[10,11] and clear PCPAR[10,11] and PCDIR[10,11,15].
3. Configure port A to enable CLK3.Set PAPAR[5] and clear PADIR[5].
4. Connect CLK3 to SCC1 using the SI. Write 0b110 to SICR[R1CS] and SICR[T1CS].
5. Connect the SCC1 to the NMSI (its own set of pins) and clear SICR[SC1].
6. Initialize the SDMA configuration register (SDCR) to 0x0001.
7. Write RBASE with 0x0000 and TBASE with 0x0008 in the SCC1 parameter RAM to point to one RxBD at the beginning of dual-port RAM followed by one TxBD.
8. Write 0x0001 to CPCR to execute the INIT RX AND TX PARAMS command for SCC1. This command updates RBPTR and TBPTR of the serial channel with the new values of RBASE and TBASE.
9. Write RFCR and TFCR with 0x10 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 0x0000_FFFF to comply with the 16-bit CRC-CCITT.
12. Write CRC_C with 0x0000_F0B8 to comply with the 16-bit CRC-CCITT.

13. Initialize the RxBD. Assume the Rx buffer is at 0x0000_1000 in main memory. Write 0xB000 to RxBD[Status and Control], 0x0000 to RxBD[Data Length] (optional), and 0x0000_1000 to RxBD[Buffer Pointer].
14. Initialize the TxBD. Assume the Tx buffer is at 0x0000_2000 in main memory and contains five 8-bit characters. Write 0xBC00 to TxBD[Status and Control], 0x0005 to TxBD[Data Length], and 0x0000_2000 to TxBD[Buffer Pointer].
15. Write 0xFFFF to SCCE to clear any previous events.
16. Write 0x0013 to SCCM to enable the TXE, TXB, and RXB interrupts.
17. Write 0x4000_0000 to the CPM interrupt mask register (CIMR) to allow SCC1 to generate a system interrupt. The CICR should also be initialized.
18. Write 0x0000_1980 to GSMR_H1 to configure the transparent channel.
19. Write 0x0000_0000 to GSMR_L1 to configure $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ to automatically control transmission and reception (DIAG bits). Normal operation of the transmit clock is used. Note that the transmitter (ENT) and receiver (ENR) are not enabled yet.
20. Write 0x0000_0030 to GSMR_L1 to enable the SCC2 transmitter and receiver. This additional write ensures that the ENT and ENR bits are enabled last.

Note that after 5 bytes are sent, the Tx buffer is closed and after 16 bytes are received the Rx buffer is closed. Any data received after 16 bytes causes a busy (out-of-buffers) condition since only one RxBD is prepared.

Chapter 29
Serial Management Controllers (SMCs)

The two serial management controllers (SMCs) are full-duplex ports that can be configured independently to support one of three protocols—UART, transparent, or general-circuit interface (GCI). Simple UART operation is used to provide a debug/monitor port in an application, which allows the SCC to be free for other purposes. The SMC in UART mode is not as complex as the SCC in UART mode. The SMC clock can be derived from one of the four internal baud rate generators (BRGs) or from an external clock pin. However, the clock should be a 16× clock.

In totally transparent mode, the SMC can be connected to TDM channel (such as a T1 line) or directly to its own set of pins. The receive and transmit clocks are derived from the TDM channel, the internal BRGs, or from an external 1× clock. The transparent protocol allows the transmitter and receiver to use the external synchronization pin. The SMC in transparent mode is not as complex as the SCC in transparent mode.

Each SMC supports the C/I and monitor channels of the GCI bus, for which the SMC connects to a time-division multiplex (TDM) channel in the serial interface (SI). Chapter 20, “Serial Interface,” describes GCI interface configuration.

The SMCs support loopback and echo modes for testing. The SMC receiver and transmitter are double-buffered, corresponding to an effective FIFO size (latency) of two characters. Figure 29-1 shows the SMC block diagram.

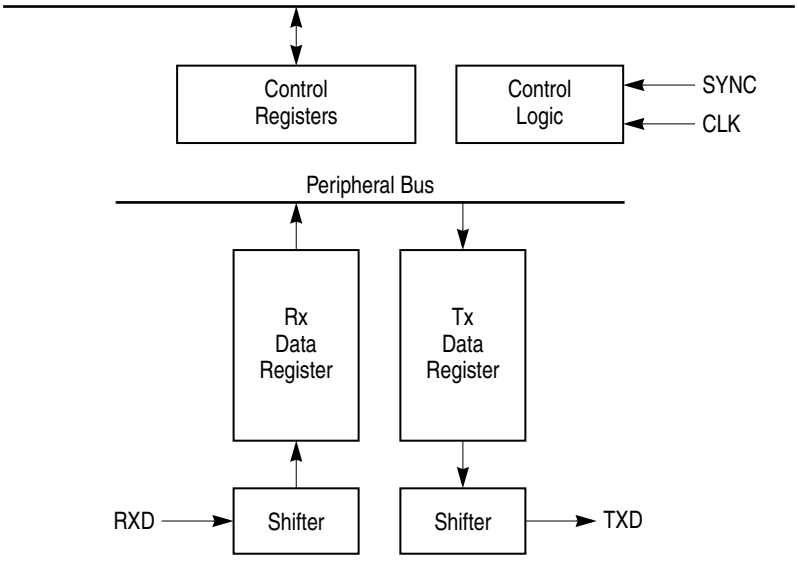


Figure 29-1. SMC Block Diagram

The receive data source can be L1RXDa if the SMC is connected to the TDM channel of the SI or SMRXD if it is connected to the NMSI. Likewise, the transmit data source can be L1TXD if using the TDM or SMTXD if using the NMSI.

If the SMC is connected to TDM, the SMC receive and transmit clocks can be independent from each other, as defined in Chapter 20, “Serial Interface.” However, if the SMC is connected to the NMSI, receive and transmit clocks must be connected to a single clock source (SMCLK), an internal signal name for a clock generated from the bank of clocks. SMCLK originates from an external pin or one of the four internal BRGs. See Section 20.3, “NMSI Configuration.”

An SMC connected to TDM derives a synchronization pulse from the TSA. An SMC connected to the NMSI using transparent protocol can use $\overline{\text{SMSYN}}$ for synchronization to determine when to start a transfer. $\overline{\text{SMSYN}}$ is not used when the SMC is in UART mode.

29.1 SMC Features

The following is a list of the SMC’s main features:

- Each SMC can implement the UART protocol on its own pins

Each SMC can implement a totally transparent protocol on a multiplexed (TDM) or nonmultiplexed (NMSI) line. The transparent mode can also be used for a fast connection between MPC855Ts.

- Each SMC channel fully supports the C/I and monitor channels of the GCI (IOM-2) in ISDN applications
- Two SMCs support the two sets of C/I and monitor channels in the SCIT channels 0 and 1
- Full-duplex operation

- Local loopback and echo capability for testing

29.2 Common SMC Settings and Configurations

The following sections describe settings and configurations that are common to the serial management controllers.

29.2.1 SMC Mode Registers (SMCMR*n*)

The two SMC mode registers (SMCMR), shown in Figure 29-2, select the SMC mode as well as mode-specific parameters. The functions of SMCMR[8–15] are the same for each protocol. SMCMR[0–7] vary according to the protocol selected by SMCMR[SM].

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field: UART	—	CLEN				SL	PEN	PM	—		SM		DM		TEN	REN
Transparent						—	BS	REVD								
GCI						ME	—	C#								
Reset	0															
R/W	R/W															
Address	0xA82 (SMCMR1), 0xA92 (SMCMR2)															

Figure 29-2. SMC Mode Registers (SMCMR*n*)

These registers are affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Table 29-1 describes SMCMR fields.

Table 29-1. SMCMR Field Descriptions

Bits	Name	Description
0	—	Reserved, should be cleared

Table 29-1. SMCMR Field Descriptions (continued)

Bits	Name	Description
1–4	CLEN	Character length (UART). Number of bits in the character minus one. The total is the sum of 1 (start bit always present) + number of data bits (5–14) + 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. Characters range from 5–14 bits. If the data bit length is less than 8, the msbs of each byte in memory are not used on transmit and are written with zeros on receive. If the length is more than 8, the msbs of each 16-bit word are not used on transmit and are written with zeros on receive. The character must not exceed 16 bits. For a 14-bit data length, set SL to one stop bit and disable parity. For a 13-bit data length with parity enabled, set SL to one stop bit. Writing values 0 to 3 to CLEN causes erratic behavior.
		Character length (transparent). The values 3–15 specify 4–16 bits per character. If a character 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. If character length is more than 8 bits but less than 16, the msbs of the half-word in buffer memory are not used on transmit and are written with zeros on receive. Note: Using values 0–2 causes erratic behavior. Larger character lengths increase an SMC channel's potential performance and lowers the performance impact of other channels. For instance, using 16- rather than 8-bit characters is encouraged if 16-bit characters are acceptable in the end application.
		Character length (GCI). Number of bits in the C/I and monitor channels of the SCIT channels 0 or 1. Values 0–15 correspond to 1–16 bits. CLEN should be 13 for SCIT channel 0 or GCI (8 data bits, plus A and E bits, plus 4 C/I bits = 14 bits). It should be 15 for the SCIT channel 1 (8 data, bits, plus A and E bits, plus 6 C/I bits = 16 bits).
5	SL	Stop length. (UART) 0 One stop bit. 1 Two stop bits.
	—	Reserved, should be cleared (transparent)
	ME	Monitor enable. (GCI) 0 The SMC does not support the monitor channel. 1 The SMC supports the monitor channel.
6	PEN	Parity enable. (UART) 0 No parity. 1 Parity is enabled for the transmitter and receiver as determined by the PM bit.
	BS	Byte sequence (transparent). For a character length greater than 8 bits, BS controls the byte transmission sequence if REVD is set. Clear BS to maintain compatibility with MC68360 QUICC. 0 Normal mode. Should be selected if the character length is not larger than 8 bits. 1 Transmit lower address byte first.
	—	Reserved, should be cleared. (GCI)
7	PM	Parity mode. (UART) 0 Odd parity. 1 Even parity.
	REVD	Reverse data. (transparent) 0 Normal mode. 1 Reverse the character bit order. The msb is sent first.
	C#	SCIT channel number. (GCI) 0 SCIT channel 0 1 SCIT channel 1. Required for Siemens ARCOFI and SGS S/T chips.
8–9	—	Reserved, should be cleared

Table 29-1. SMCMR Field Descriptions (continued)

Bits	Name	Description
10–11	SM	SMC mode. 00 GCI or SCIT support. 01 Reserved. 10 UART (must be selected for SMC UART operation). 11 Totally transparent operation.
12–13	DM	Diagnostic mode. 00 Normal operation. 01 Local loopback mode. 10 Echo mode. 11 Reserved.
14	TEN	SMC transmit enable. 0 SMC transmitter disabled. 1 SMC transmitter enabled.
15	REN	SMC receive enable. 0 SMC receiver disabled. 1 SMC receiver enabled.

29.2.2 SMC Buffer Descriptors (BDs)

In UART and transparent modes, the SMC's memory structure is like the SCC's in that SMC-associated data is stored in buffers. Each buffer is referenced by a BD and organized in a BD table located in the dual-port RAM. See Figure 29-3.

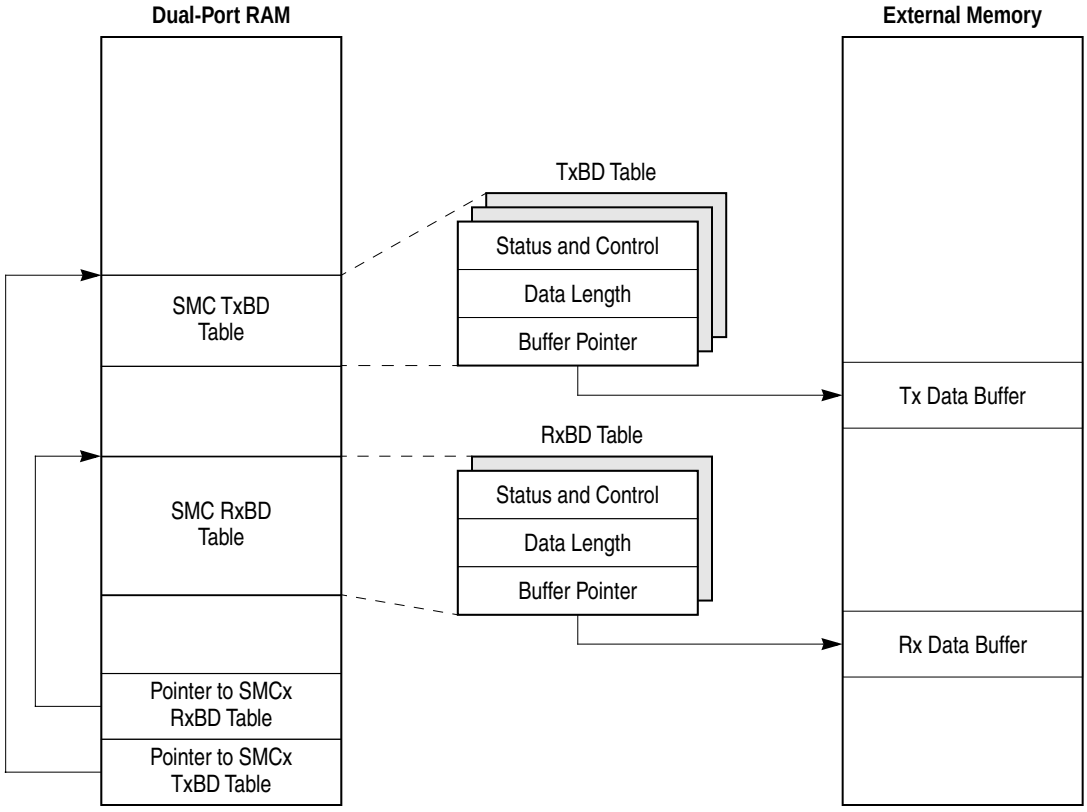


Figure 29-3. SMC Memory Structure

The BD table allows buffers to be defined for transmission and reception. Each table forms a circular queue. The CP uses BDs to confirm reception and transmission so that the processor knows buffers have been serviced. The data resides in external or internal buffers.

When SMCs 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, see Section 29.5, “SMC in GCI Mode.”

29.2.3 SMC Parameter RAM

Each SMC parameter RAM area begins at the same offset from each SMC base. The protocol-specific portions of the SMC parameter RAM are discussed in the sections that follow. The SMC parameter RAM shared by the UART and transparent protocols is shown in Table 29-2. Parameter RAM for GCI protocol is described in Section 29.5.1, “SMC GCI Parameter RAM.”

Table 29-2. SMC UART and Transparent Parameter RAM Memory Map

Offset [1]	Name	Width	Description
0x00	**RBASE**	Hword	RxBDs and TxBDs base address. (BD table pointer) Define starting points in the dual-port RAM of the set of BDs for the SMC send and receive functions. They allow flexible partitioning of the BDs. By selecting RBASE and TBASE entries for all SMCs and by setting W in the last BD in each list, BDs are allocated for the send and receive side of every SMC. Initialize these entries before enabling the corresponding channel. Configuring BD tables of two enabled SMCs to overlap causes erratic operation. RBASE and TBASE should be a multiple of eight.
0x02	**TBASE**	Hword	
0x04	**RFCR**	Byte	Rx/Tx function code. See Section 29.2.3.1, “SMC Function Code Registers (RFCR/TFCR).”
0x05	**TFCR**	Byte	
0x06	**MRBLR**	Hword	Maximum receive buffer length. The most bytes the MPC855T writes to a Rx buffer before moving to the next buffer. It can write fewer bytes than MRBLR if a condition like an error or end-of-frame occurs, but it cannot exceed MRBLR. Rx buffers should not be smaller than MRBLR. SMC Tx buffers are unaffected by MRBLR. Tx buffers can be individually given varying lengths through the data length field. MRBLR can be changed while an SMC is operating only if it is done in a single bus cycle with one 16-bit move (not two 8-bit bus cycles back-to-back). This occurs when the CP shifts control to the next RxBD, so the change does not take effect immediately. To guarantee the exact RxBD on which the change occurs, change MRBLR only while the SMC receiver is disabled. MRBLR should be greater than zero and should be even if character length exceeds 8 bits.
0x08	RSTATE	Word	Rx internal state. Can be used only by the CP.
0x0C	—	Word	Rx internal data pointer. [2] Updated by the SDMA channels to show the next address in the buffer to be accessed.
0x10	RBPTR	Hword	RxBD pointer. Points to the next BD for each SMC channel that the receiver transfers data to when it is in idle state, or to the current BD during frame processing. After a reset or when the end of the BD table is reached, the CP initializes RBPTR to the value in RBASE. Most applications never need to write RBPTR, but it can be written when the receiver is disabled or when no receive buffer is in use.
0x12	—	Hword	Rx internal byte count. [2] A down-count value initialized with the MRBLR value and decremented with every byte the SDMA channels write.
0x14	—	Word	Rx temp. [2] Can be used only by the CP.
0x18	TSTATE	Word	Tx internal state. Can be used only by the CP.
0x1C	—	Word	Tx internal data pointer. [2] Updated by the SDMA channels to show the next address in the buffer to be accessed.
0x20	TBPTR	Hword	TxBD pointer. Points to the next BD for each SMC channel the transmitter transfers data from when it is in idle state or to the current BD during frame transmission. After reset or when the end of the table is reached, the CP initializes TBPTR to the TBASE value. Most applications never need to write TBPTR, but it can be written when the transmitter is disabled or when no transmit buffer is in use. For instance, after a STOP TRANSMIT or GRACEFUL STOP TRANSMIT command is issued and the frame completes its transmission.
0x22	—	Hword	Tx internal byte count. [2] A down-count value initialized with the TxBD data length and decremented with every byte the SDMA channels read.
0x24	—	Word	Tx temp. [2] Can be used only by the CP.

Table 29-2. SMC UART and Transparent Parameter RAM Memory Map (continued)

Offset [1]	Name	Width	Description
0x28	—	Hword	First half-word of protocol-specific area.
0x32	—	Hword	Last half-word of protocol-specific area.

1 From SMC base address. SMC base = IMMR + 3E80 (SMC1), 3F80 (SMC2).
2 Not accessed for normal operation. May hold helpful information for experienced users and for debugging.

To extract data from a partially full Rx buffer, issue a CLOSE RXBD command.

Certain parameter RAM values must be initialized before the SMC is enabled. Other values are initialized or written by the CP. Once values are initialized, software typically does not need to update them because activity centers mostly around Tx and Rx BDs rather than parameter RAM. However, note the following:

- Parameter RAM can be read at any time.
- Values that pertain to the SMC transmitter can be written only if SMCMR[TEN] is zero or between the STOP TRANSMIT and RESTART TRANSMIT commands.
- Values for the SMC receiver can be written only when SMCMR[REN] is zero, or, if the receiver is previously enabled, after an ENTER HUNT MODE command is issued but before the CLOSE RXBD command is issued and REN is set.

29.2.3.1 SMC Function Code Registers (RFCR/TFCR)

Each SMC channel has two function code registers—one for receiving (RFCR*n*) and one for transmitting (TFCR*n*). The function code entry contains the value to appear on the function code pins AT[1–3] when the associated SDMA channel accesses memory. The FCRs also control byte-ordering. See Figure 29-4.

Bit	0	1	2	3	4	5	6	7
R/W	R/W							
Address	SMC base + 0x04 (RFCR)/SMC base + 0x05 (TFCR)							

Figure 29-4. SMC Function Code Registers (RFCR/TFCR)

Table 29-3 describes RFCR fields.

Table 29-3. RFCR/TFCR Field Descriptions

Bit	Name	Description
0–2	—	Reserved, should be cleared.

Table 29-3. RFCR/TFCR Field Descriptions (continued)

Bit	Name	Description
3–4	**BO**	Byte ordering. Set BO to select the required byte ordering for the buffer. If BO is changed on-the-fly, it takes effect at the beginning of the next frame (Ethernet, HDLC, and transparent) or at the beginning of the next BD. See Appendix A, “Byte Ordering.” 00 Reserved 01 Modified little-endian. 1x Big-endian or true little-endian.
5–7	**AT[1–3]**	Address type 1–3. Contains the user-defined function code value used during the SDMA channel memory access. AT[0] is always driven high to identify this channel access as a DMA-type access.

29.2.4 Disabling SMCs On-the-Fly

An SMC can be disabled and reenabled later by ensuring that buffers are closed properly and new data is transferred to or from a new buffer. Such a sequence is required if the parameters to be changed are not dynamic. If the register or bit description states that on-the-fly changes are allowed, the sequences need not be followed and the register or bits may be changed directly.

Note that the SMC does not have to be fully disabled for parameter RAM to be modified. Table 29-2 describes when parameter RAM values can be modified. To disable the SCC, SMCs, SPI, and the I^2C, use CPCR[RST] to reset the CPM.

29.2.4.1 SMC Transmitter Full Sequence

Follow these steps to fully enable or disable the SMC transmitter:

1. If the SMC is sending data, issue a STOP TRANSMIT command to stop transmission smoothly. If the SMC is not sending, this command is not required.
2. Clear SMCMR[TEN] to disable the SMC transmitter and put it in reset state.
3. Update SMC transmit parameters, including the parameter RAM. To switch protocols or reinitialize parameters, issue an INIT TX PARAMETERS command.
4. Issue a RESTART TRANSMIT if an INIT TX PARAMETERS was not issued in step 3.
5. Set SMCMR[TEN]. Transmission now begins using the TxBD that the TBPTR value points to as soon as the R bit is set in that TxBD.

29.2.4.2 SMC Transmitter Shortcut Sequence

This shorter sequence reinitializes transmit parameters to the state they had after reset.

1. Clear SMCMR[TEN].
2. Make any changes, then issue an INIT TX PARAMETERS command.
3. Set SMCMR[TEN].

29.2.4.3 SMC Receiver Full Sequence

Follow these steps to fully enable or disable the receiver:

1. Clear SMCMR[REN]. Reception is aborted immediately, which disables the SMC receiver and puts it in a reset state.
2. Modify SMC receive parameters, including parameter RAM. To switch protocols or reinitialize SMC receive parameters, issue an INIT RX PARAMETERS command.
3. Issue a CLOSE RXBD command if INIT RX PARAMETERS was not issued in step 2.
4. Set SMCMR[REN]. Reception immediately uses the RxBD that RBPTR points to if E is set in that RxBD.

29.2.4.4 SMC Receiver Shortcut Sequence

This shorter sequence reinitializes receive parameters to their state after reset.

1. Clear SMCMR[REN].
2. Make any changes, then issue an INIT RX PARAMETERS command.
3. Set SMCMR[REN].

29.2.4.5 Changing SMC Protocols

To switch the protocol that the SMC is executing without resetting the board or affecting the other SMC, follow these steps:

1. Clear SMCMR[REN, TEN].
2. Make any SMCMR changes, modify the parameter RAM appropriately, and issue an INIT TX AND RX PARAMETERS COMMAND to initialize transmit and receive parameters.
3. Set SMCMR[REN, TEN]. The SMC is now enabled with the new protocol.

29.2.5 Saving Power

When the SMCMR[TEN, REN] are zero, the SMC consumes very little power.

29.2.6 Handling Interrupts in the SMC

Follow these steps to handle an interrupt in the SMC:

1. Once an interrupt occurs, read SMCE to identify the interrupt source. The SMCE bits are usually cleared at this time.
2. Process the TxBD to reuse it if SMCE[TX] is set. Extract data from the RxBD if SMCE[RX] is set. To send another buffer, set R in the TxBD.
3. Clear CISR[SMC1].
4. Execute the **rfi** instruction.

29.3 SMC in UART Mode

SMCs generally offer less functionality and performance in UART mode than SCC, which makes them more suitable for simpler debug/monitor ports instead of full-featured UARTs. SMCs do not support the following features in UART mode:

- $\overline{\text{RTS}}$, $\overline{\text{CTS}}$, and $\overline{\text{CD}}$ signals
- 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) (That is, a 16× clock is required.)
- Interrupts on special control character reception
- Ability to transmit data on demand using the TODR
- SCCS register to determine idle status of the receive pin
- Other features for the SCC as described in the GSMR

However, the SMC UART frame format, shown in Figure 29-5, allows a data length of up to 14 bits. The SCC format supports only up to 8 bits.

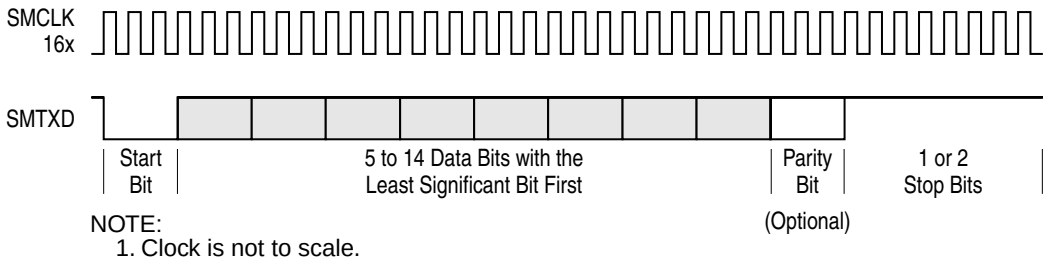


Figure 29-5. SMC UART Frame Format

29.3.1 SMC UART 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

29.3.2 SMC UART-Specific Parameter RAM

For UART mode, the protocol-specific area of the SMC parameter RAM is mapped as in Table 29-4.

Table 29-4. SMC UART-Specific Parameter RAM Memory Map

Offset [1]	Name	Width	Description
0x28	**MAX_IDL**	Hword	Maximum idle characters. When a character is received on the line, the SMC starts counting idle characters received. If MAX_IDL idle characters arrive before the next character, an idle time-out occurs and the buffer closes, which sends an interrupt request to the core to receive data from the buffer. An idle character is defined as a full character length of logic high. MAX_IDL can be used to demarcate frames in UART mode. Clearing MAX_IDL disables this function so idle never causes the buffer to close, regardless of how many idle characters are received. The length of an idle character is calculated as follows: 1 + data length (5 to 14) + 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, character length is 10 bits.
0x2A	IDLC	Hword	Temporary idle counter. Down-counter in which the CP stores the current idle counter value in the MAX_IDL time-out process.
0x2C	BRKLN	Hword	Last received break length. Holds the length of the last received break character sequence measured in character units. For example, if the receive signal is low for 20 bit times and the defined character length is 10 bits, BRKLN = 0x002, indicating that the break sequence is at least 2 characters long. BRKLN is accurate to within one character length.
0x2E	**BRKEC**	Hword	Receive break condition counter. Counts break conditions on the line. A break condition may last for hundreds of bit times, yet BRKEC increments only once during that period.
0x30	**BRKCR**	Hword	Break count register (transmit). Determines the number of break characters the UART controller sends when the SMC sends a break character sequence after a STOP TRANSMIT command. For 8 data bits, no parity, 1 stop bit, and 1 start bit, each break character is 10 zeros.
0x32	R_MASK	Hword	Temporary bit mask.

[1] From SMC base address. SMC base = IMMR + 0x3E80 (SMC1), 0x3F80 (SMC2).

29.3.3 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 sending idles, which are defined as the full character length of logic high. The SMC immediately polls the first BD in the transmit channel BD table and once every character time after that, depending on character length. When there is a message to transmit, the SMC fetches data from memory and starts sending the message.

When a BD data is completely written to the transmit FIFO, the SMC writes the message status bits into the BD and clears R. An interrupt is issued if the I bit in the BD is set. If the next TxBD is ready, the data from its buffer is appended to the previous data and sent over the transmit pin without any gaps between buffers. If the next TxBD is not ready, the SMC starts sending idles and waits for the next TxBD to be ready.

By appropriately setting the I bit in each BD, interrupts can be generated after each buffer, a specific buffer, or each block is sent. The SMC then proceeds to the next BD. If the CM bit is set in the TxBD, the R bit is not cleared, allowing a buffer to be automatically resent next time the CP accesses this buffer. For instance, if a single TxBD is initialized with the CM and W bits set, the buffer is sent continuously until R is cleared in the BD.

29.3.4 SMC UART Channel Reception Process

When the core enables the SMC receiver, it enters HUNT mode and waits for the first character. The CP then checks the first RxBD to see if it is empty and starts storing characters in the buffer. When the buffer is full or the MAX_IDL timer expires (if enabled), the SMC clears the E bit in the BD and generates an interrupt if the I bit in the BD is set. If incoming data exceeds the buffer's length, the SMC fetches the next BD, and, if it is empty, continues transferring data to this BD's buffer. If CM is set in the RxBD, the E bit is not cleared, so the CP can overwrite this buffer on its next access.

29.3.5 Data Handling Modes: Character- and Message-Oriented

UART mode uses the same data structures as other modes. The structures support multibuffer operation and allow break and preamble sequences to be sent. Overrun, parity, and framing errors are reported via the BDs. At its simplest, the SMC UART controller functions in a character-oriented environment, whereas each character is sent with the selected stop bits and parity. They are received into separate 1-byte buffers. A maskable interrupt can be generated when each buffer is received.

Many applications can take advantage of the message-oriented capabilities that the SMC UART supports through linked buffers for sending or receiving. Data is handled in a message-oriented environment, so entire messages can be handled instead of individual characters. A message can span several linked buffers; each one can be sent and received as a linked list of buffers without core intervention, which simplifies programming and saves processor overhead. In a message-oriented environment, an 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.

29.3.6 SMC UART Commands

Table 29-5 describes transmit commands issued to the CPCR.

Table 29-5. Transmit Commands

Command	Description
STOP TRANSMIT	Disables transmission of characters on the transmit channel. If the SMC UART controller receives this command while sending a message, it stops sending. The SMC UART controller finishes sending any data that has already been sent to its FIFO and shift register and then stops sending data. The TBPTR is not advanced when this command is issued. The SMC UART controller sends a programmable number of break sequences and then sends idles. The number of break sequences, which can be zero, should be written to the BRKCR before this command is issued to the SMC UART controller.
RESTART TRANSMIT	Enables characters to be sent 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 TxBD table.
INIT TX PARAMETERS	Initializes 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 PARAMETERS command can also be used to reset the transmit and receive parameters.

Table 29-6 describes receive commands issued to the CPCR.

Table 29-6. Receive Commands

Command	Description
ENTER HUNT MODE	Use the CLOSE RXBD command instead of ENTER HUNT MODE for an SMC UART channel.
CLOSE RXBD	Forces the SMC to close the current RxBD if it is currently being used and to use the next BD in the list for any subsequently received data. If the SMC is not receiving data, no action is taken.
INIT RX PARAMETERS	Initializes receive parameters in this serial channel parameter RAM to reset state. Issue it only if the receiver is disabled. INIT TX AND RX PARAMETERS resets both receive and transmit parameters.

29.3.7 Sending a Break

A break is an all-zeros character without stop bits. It is sent by issuing a STOP TRANSMIT command. After sending any outstanding data, the SMC sends a character of consecutive zeros, the number of which is the sum of the character length, plus the number of start, parity, and stop bits. The SMC sends a programmable number of break characters according to BRKCR and then reverts to idle or sends data if a RESTART TRANSMIT is issued before completion. When the break completes, the transmitter sends at least one idle character before sending any data to guarantee recognition of a valid start bit.

29.3.8 Sending a Preamble

A preamble sequence provides a way to ensure that the line is idle before a new message transfer begins. The length of the preamble sequence is constructed of consecutive ones that are one character long. If the preamble bit in a BD is set, the SMC sends a preamble sequence before sending that 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 sent on the transmit pin with no delay between them.

29.3.9 Handling Errors in the SMC UART Controller

The SMC UART controller reports character reception errors via the channel RxBD status fields and the SMC event register (SMCE). Table 29-7 shows the possible UART receiving errors. The SMC UART controller has no transmission errors.

Table 29-7. SMC UART Errors

Error	Description
Overrun	The SMC maintains a two-character length FIFO for receiving data. Data is moved to the buffer after the first character is received into the FIFO; if a receiver FIFO overrun occurs, the channel writes the received character into the internal FIFO. It then writes the character to the buffer, closes it, sets RxBD[OV], and generates the RX interrupt if it is enabled. Reception then resumes as normal. Overrun errors that occasionally occur when the line is idle can be ignored.
Parity	The channel writes the received character to the buffer, closes it, sets the PR bit in the BD, and generates the RX interrupt if it is enabled. Reception then resumes as normal.
Idle Sequence Receive	An idle is found when a character of all ones is received, at which point the channel counts consecutive idle characters. If the count reaches MAX_IDL, the buffer is closed and an RX interrupt is generated. If no receive buffer is open, this does not generate an interrupt or any status information. The idle counter is reset each time a character is received.
Framing	The SMC received a character with no stop bit. When it occurs, the channel writes the received character to the buffer, closes the buffer, sets FR in the BD, and generates the RX interrupt if it is enabled. When this error occurs, parity is not checked for the character.
Break Sequence	The SMC receiver received an all-zero character with a framing error. The channel increments BRKEC, generates a maskable BRK interrupt in SMCE, measures the length of the break sequence, and stores this value in BRKLN. If the channel was processing a buffer when the break was received, the buffer is closed with the BR bit in the RxBD set. The RX interrupt is generated if it is enabled.

29.3.10 SMC UART Receive BD (RxBD)

The CP reports information about the received data in each buffer's RxBD, shown in Figure 29-6. The CP then closes the current buffer, generates a maskable interrupt, and starts receiving data into the next buffer after one of the following occurs:

- An error is received during message processing
- A full receive buffer is detected

- A programmable number of consecutive idle characters are received

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**E**	—	**W**	**I**	—		**CM**	ID	—		BR	FR	PR	—	OV	—
Offset + 2	Data Length															
Offset + 4	**Rx Buffer Pointer**															
Offset + 6																

Figure 29-6. SMC UART Receive BD (RxBD)

Table 29-8 describes SMC UART RxBD status and control fields.

Table 29-8. SMC UART RxBD Status and Control Field Descriptions

Bit	Name	Description
0	**E**	Empty. 0 The buffer is full or data reception stopped due to an error. The core can read or write any fields of this RxBD. The CP does not use this BD while E is zero. 1 The buffer is empty or reception is in progress. This RxBD and its buffer are owned by the CP. Once E is set, the core should not write any fields of this RxBD.
1	—	Reserved, should be cleared
2	**W**	Wrap (last BD in RxBD table). 0 Not the last BD in the table. 1 Last BD in the table. After this buffer is used, the CP receives incoming data into the first BD that RBASE points to in the table. The number of RxBDs in this table is determined only by the W bit.
3	**I**	Interrupt. 0 No interrupt is generated after this buffer is filled. 1 The SMCE[RX] is set when this buffer is completely filled by the CP, indicating the need for the core to process the buffer. RX can cause an interrupt if it is enabled.
4–5	—	Reserved, should be cleared
6	**CM**	Continuous mode. 0 Normal operation. 1 The CP does not clear the E bit after this BD is closed, allowing the CP to automatically overwrite the buffer when it next accesses the BD. However, E is cleared if an error occurs during reception, regardless of how CM is set.
7	ID	Buffer closed on reception of idles. Set when the buffer has closed because a programmable number of consecutive idle sequences is received. The CP writes ID after received data is in the buffer.
8–9	—	Reserved, should be cleared
10	BR	Buffer closed on reception of break. Set when the buffer closes because a break sequence was received. The CP writes BR after the received data is in the buffer.
11	FR	Framing error. Set when a character with a framing error is received and located in the last byte of this buffer. A framing error is a character with no stop bit. A new receive buffer is used to receive additional data. The CP writes FR after the received data is in the buffer.
12	PR	Parity error. Set when a character with a parity error is received in the last byte of the buffer. A new buffer is used for additional data. The CP writes PR after received data is in the buffer.
13	—	Reserved, should be cleared

Table 29-8. SMC UART RxBD Status and Control Field Descriptions (continued)

Bit	Name	Description
14	OV	Overrun. Set when a receiver overrun occurs during reception. The CP writes OV after the received data is in the buffer.
15	—	Reserved, should be cleared

Data length represents the number of octets the CP writes into the buffer. After data is received in the buffer, the CP only writes the data length once as the BD closes. Note that the memory allocated for this buffer should be no smaller than MRBLR. The Rx buffer pointer points to the first location of the buffer and must be even. The buffer can be in internal or external memory. Figure 29-7 shows an example of how RxBDs are used in receiving 10 characters, an idle period, and five characters (one with a framing error). The example assumes that MRBLR = 8.

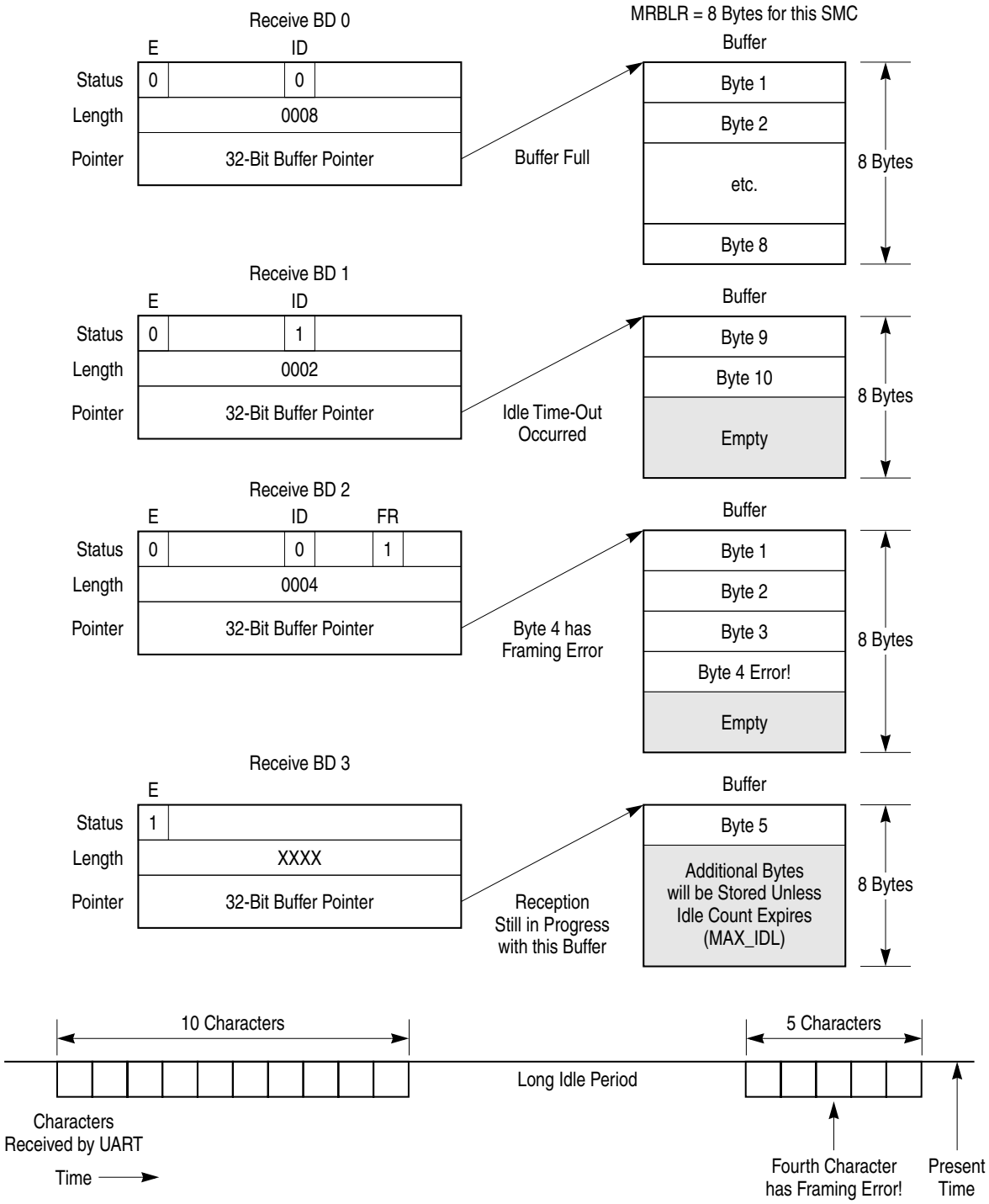


Figure 29-7. SMC UART Receiving using RxBDs

29.3.11 SMC UART Transmit BD (TxBD)

Data is sent to the CPM for transmission on an SMC channel by arranging it in buffers referenced by descriptors in the channel's TxBD table. Using the BDs, the CP 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**	—	**W**	**I**	—		**CM**	**P**	—							
Offset + 2	**Data Length**															
Offset + 4	**Tx Buffer Pointer**															
Offset + 6																

Figure 29-8. SMC UART Transmit BD (TxBD)

Table 29-9 describes SMC UART TxBD status and control fields.

Table 29-9. SMC UART TxBD Status and Control Field Descriptions

Bits	Name	Description
0	**R**	Ready 0 The buffer is not ready for transmission; BD and its buffer can be altered. The CP clears R after the buffer has been sent or an error occurs. 1 The buffer has not been completely sent. This BD must not be updated while R is set.
1	—	Reserved, should be cleared.
2	**W**	Wrap (final BD in the TxBD table) 0 Not the last BD in the table. 1 Last BD in the table. After this buffer is used, the CP transmits outgoing data from the first BD that TBASE points to. The number of TxBDs in this table is determined only by the W bit.
3	**I**	Interrupt 0 No interrupt is generated after this buffer is serviced. 1 The SMCE[TX] is set when this buffer is serviced. TX can cause an interrupt if it is enabled.
4–5	—	Reserved, should be cleared.
6	**CM**	Continuous mode 0 Normal operation. 1 The CP does not clear R after this BD is closed and automatically retransmits the buffer when it accesses this BD next.
7	**P**	Preamble 0 No preamble sequence is sent. 1 The UART 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 BD is zero, only a preamble is sent.
8–15	—	Reserved, should be cleared.

Data length represents the number of octets that the CP should transmit from this BD's buffer. It is never modified by the CP and normally is greater than zero. It can be zero if P is set, in which case only a preamble is sent. If there are more than 8 bits in the UART

character, data length should be even. For example, to transmit three UART characters of 8-bit data, 1 start, and 1 stop, initialize the data length field to 3. To send three UART characters of 9-bit data, 1 start, and 1 stop, the data length field should 6, because the three 9-bit data fields occupy three half words in memory (the 9 LSBs of each half word).

Tx buffer pointer points to the first location of the buffer. It can be even or odd, unless the number of data bits in the UART character is greater than 8 bits, in which case the buffer pointer must be 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.

29.3.12 SMC UART Event Register (SMCE)/Mask Register (SMCM)

The SMC event register (SMCE) generates interrupts and report events recognized by the SMC UART channel. When an event is recognized, the SMC UART controller sets the corresponding SMCE bit. SMCE bits are cleared by writing ones; writing zeros has no effect. The SMC mask register (SMCM) has the same bit format as SMCE. Setting an SMCM bit enables, and clearing it disables, the corresponding interrupt. All unmasked bits must be cleared before the CP clears the internal interrupt request.

Bit	0	1	2	3	4	5	6	7
Field	—	BRKE	—	BRK	—	BSY	TX	RX
Reset	0							
R/W	R/W							
Address	0xA86 (SMCE1), 0xA96 (SMCE2)/ 0xA8A (SMCM1), 0xA9A (SMCM2)							

Figure 29-9. SMC UART Event Register (SMCE)/Mask Register (SMCM)

These registers are affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Table 29-10 describes SMCE/SMCM fields.

Table 29-10. SMCE/SMCM Field Descriptions

Bits	Name	Description
0	—	Reserved, should be cleared.
1	BRKE	Break end. Set no sooner than after one idle bit is received after the break sequence.
2	—	Reserved, should be cleared.
3	BRK	Break character received. Set when a break character is received. If a very long break sequence occurs, this interrupt occurs only once after the first all-zeros character is received.
4	—	Reserved, should be cleared.
5	BSY	Busy condition. Set when a character is received and discarded due to a lack of buffers. Set no sooner than the middle of the last stop bit of the first receive character for which there is no available buffer. Reception resumes when an empty buffer is provided.

Table 29-10. SMCE/SMCM Field Descriptions (continued)

Bits	Name	Description
6	TX	Tx buffer. Set when the transmit data of the last character in the buffer is written to the transmit FIFO. Wait two character times to ensure that data is completely sent over the transmit pin.
7	RX	Rx buffer. Set when a buffer is received and its associated RxBD is closed. Set no sooner than the middle of the last stop bit of the last character that is written to the receive buffer.

Figure 29-10 shows an example of the timing of various events in the SMCE.

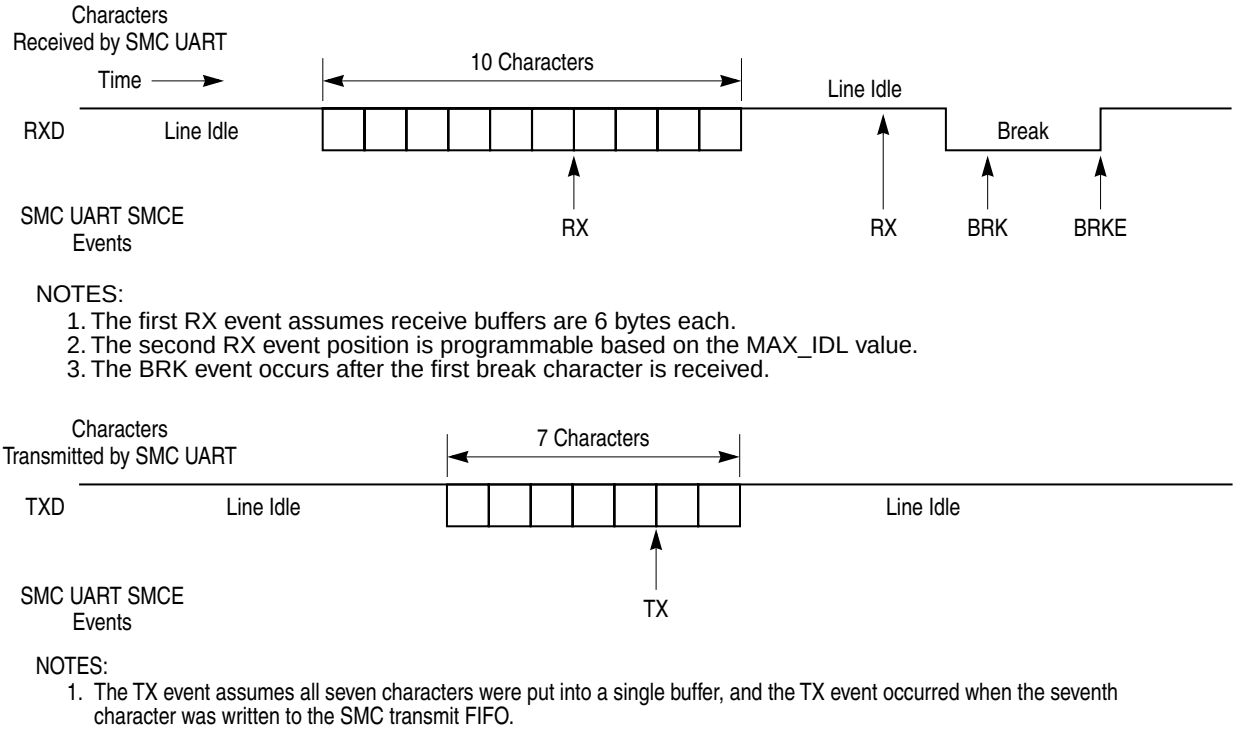


Figure 29-10. SMC UART Interrupts Example

29.3.13 SMC UART Controller Programming Example

The following initialization sequence assumes 9,600 baud, 8 data bits, no parity, and 1 stop bit in a 25-MHz system. BRG1 and SMC1 are used.

1. Configure the port B pins to enable SMTXD1 and SMRXD1. Set PBPAR[24, 25] then clear PBDIR[24, 25] and PBODR[24, 25].
2. Configure the BRG1. Write BRGC1 with 0x01_0144. The DIV16 bit is not used and the divider is 162 (decimal). The resulting BRG1 clock is 16× the preferred bit rate.
3. Connect BRG1 to SMC1 using the SI. Clear SIMODE[SMC1, SMC1CS].
4. Assuming one RxBD at the beginning of dual-port RAM followed by one TxBD, write RBASE with 0x0000 and TBASE with 0x0008.

5. Write 0x0091 to CPCR to execute the INIT RX AND TX PARAMETERS command.
6. Initialize the SDMA configuration register (SDCR) to 0x0001.
7. Write RFCR and TFCR with 0x10 for normal operation.
8. Write MRBLR with the maximum number of bytes per receive buffer. Assume 16 bytes, so MRBLR = 0x0010.
9. Write MAX_IDL with 0x0000 in the SMC UART-specific parameter RAM to disable the MAX_IDL functionality for this example.
10. Clear BRKLN and BRKEC in the SMC UART-specific parameter RAM.
11. Set BRKCR to 0x0001; if a STOP TRANSMIT COMMAND is issued, one break character is sent.
12. Initialize the RxBD. Assume the Rx buffer is at 0x0000_1000 in main memory. Write 0xB000 to Rx_BD_Status, 0x0000 to Rx_BD_Length (not required), and 0x0000_1000 to Rx_BD_Pointer.
13. Assuming the Tx 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.
14. Write 0xFF to the SMCE register to clear any previous events.
15. Write 0x17 to the SMCM register to enable all possible SMC interrupts.
16. Write 0x00000010 to the CIMR so the SMC1 can generate a system interrupt. Initialize the CICR.
17. Write 0x4820 to SMCMR to configure normal operation (not loopback), 8-bit characters, no parity, 1 stop bit. The transmitter and receiver are not yet enabled.
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.

After 5 bytes are sent, the TxBD is closed. The receive buffer closes after receiving 16 bytes. Subsequent data causes a busy (out-of-buffers) condition since only one RxBD is ready.

29.4 SMC in Transparent Mode

Compared to the SCC in transparent mode, the SMCs generally have less functionality, simpler functions and slower speeds. Transparent mode is selected by setting SMCMR[SM] to 0b11. Section 29.2.1, “SMC Mode Registers (SMCMRn),” describes other protocol-specific bits in the SMCMR. The SMC in transparent mode does not support the following features:

- Independent transmit and receive clocks, unless connected to TDM channel of the SI
- CRC generation and checking
- Full $\overline{\text{RTS}}$, $\overline{\text{CTS}}$, and $\overline{\text{CD}}$ signals (supports only one $\overline{\text{SMSYN}}$ signal)

- 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

However, the SMC in transparent mode provides a data character length option of 4 to 16 bits, whereas the SCC provide 8 or 32 bits, depending on GSMR[RFW]. The SMC in transparent mode is also referred to as the SMC transparent controller.

29.4.1 SMC Transparent Mode Features

The following list summarizes the features of the SMC in transparent mode:

- Flexible buffers
- Can connect to a TDM bus using the TSA in the SI
- Can transmit and receive 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

29.4.2 SMC Transparent-Specific Parameter RAM

The protocol-specific parameter RAM for the SMC in transparent mode is reserved. Only the general SMC parameter RAM is used. See Section 29.2.3, “SMC Parameter RAM.”

29.4.3 SMC Transparent Channel Transmission Process

The transparent transmitter is designed to work with almost no core intervention. When the core enables the SMC transmitter in transparent mode, it starts sending idles. The SMC immediately polls the first BD in the transmit channel BD table and once every character time, depending on the character length (every 4 to 16 serial clocks). When there is a message to transmit, the SMC fetches the data from memory and starts sending the message when 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 before it starts transmitting. Data is sent only during the time slots defined by the TSA. Secondly, when working with its own set of pins, the transmitter starts sending when $\overline{\text{SMSYN}x}$ is asserted.

When a BD data is completely written to the transmit FIFO, the L bit is checked and if it is set, the SMC writes the message status bits into the BD and clears the R bit. It then starts transmitting idles. When the end of the current BD is reached and the L bit is not set, only

R is cleared. In both cases, an interrupt is issued according to the I bit in the BD. By appropriately setting the I bit in each BD, interrupts can be generated after each buffer, a specific buffer, or each block is sent. The SMC then proceeds to the next BD. If no additional buffers have been presented to the SMC for transmission and the L bit was cleared, an underrun is detected and the SMC begins sending idles.

If the CM bit is set in the TxBD, the R bit is not cleared, so the CP can overwrite the buffer on its next access. For instance, if a single TxBD is initialized with the CM and W bits set, the buffer is sent continuously until R is cleared in the BD.

29.4.4 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 RxBD in the table. Synchronization can be achieved in two ways. First, when the receiver is connected to 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 received only during the time slots defined by the TSA. Secondly, when working with its own set of pins, the receiver starts reception when $\overline{\text{SMSYN}x}$ is asserted.

When the buffer full, the SMC clears the E bit in the BD and generates an interrupt if the I bit in the BD is set. If incoming data exceeds the buffer length, the SMC fetches the next BD; if it is empty, the SMC continues transferring data to this BD's buffer. If the CM bit is set in the RxBD, the E bit is not cleared, so the CP can automatically overwrite the buffer on its next access.

29.4.5 Using $\overline{\text{SMSYN}}$ for Synchronization

The $\overline{\text{SMSYN}}$ signal offers a way to externally synchronize the SMC channel. This method differs somewhat from the synchronization options available in the SCC and should be studied carefully. See Figure 29-11 for an example.

Once SMCMR[REN] is set, the first rising edge of SMCLK that finds $\overline{\text{SMSYN}}$ low causes the SMC receiver to achieve synchronization. Data starts being received or latched on the same rising edge of SMCLK that latched $\overline{\text{SMSYN}}$. This is the first bit of data received. The receiver does not lose synchronization again, regardless of the state of $\overline{\text{SMSYN}}$, until REN is cleared.

Once SMCMR[TEN] is set, the first rising edge of SMCLK that finds $\overline{\text{SMSYN}}$ low synchronizes the SMC transmitter which begins sending ones asynchronously from the falling edge of $\overline{\text{SMSYN}}$. After one character of ones is sent, if the transmit FIFO is loaded (the TxBD is ready with data), data starts being send on the next falling edge of SMCLK after one character of ones is sent. If the transmit FIFO is loaded later, data starts being sent after some multiple number of all-ones characters is sent.

Note that regardless of whether the transmitter or receiver uses $\overline{\text{SMSYN}}$, it must make glitch-free transitions from high-to-low or low-to-high. Glitches on $\overline{\text{SMSYN}}$ can cause erratic behavior of the SMC.

The transmitter and receiver never lose synchronization again, regardless of the state of $\overline{\text{SMSYN}}$, until the TEN bit is cleared or an ENTER HUNT MODE command is issued.

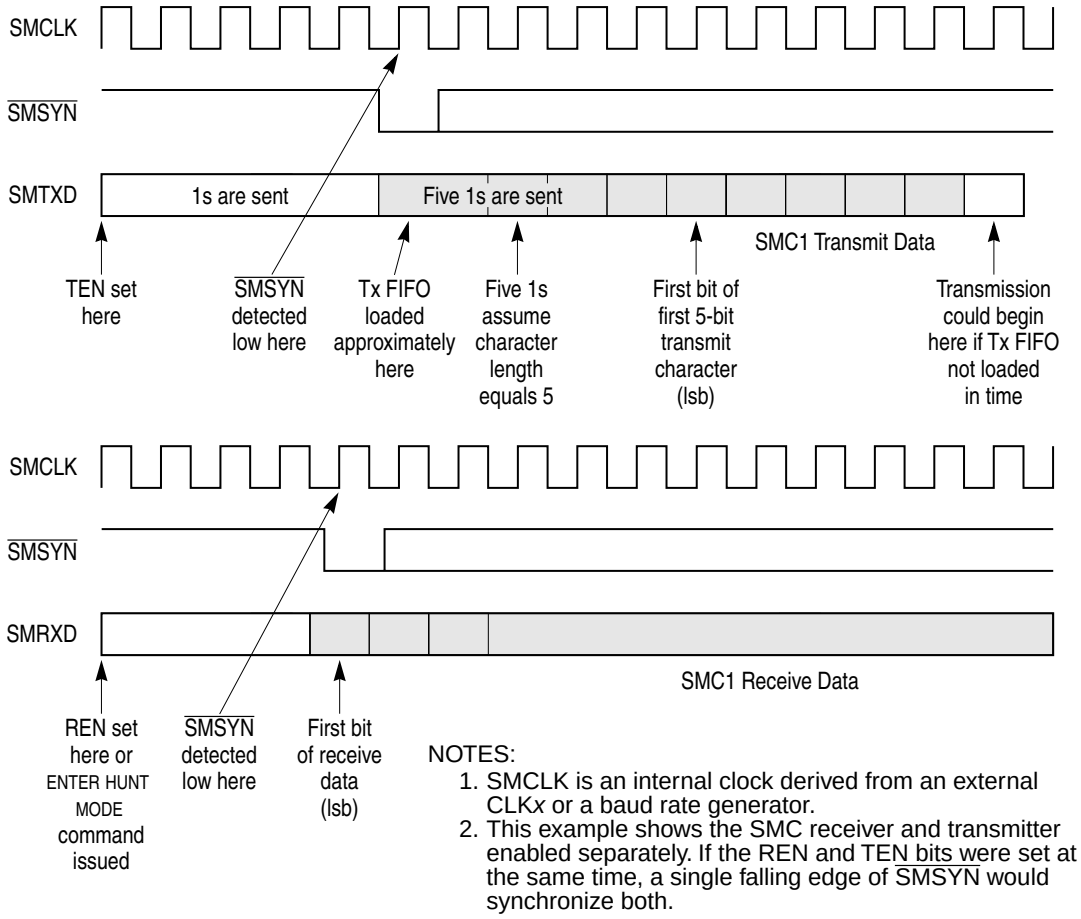


Figure 29-11. Synchronization with $\overline{\text{SMSYN}}x$

If both SMCMR[REN] and SMCMR[TEN] are set, the first falling edge of $\overline{\text{SMSYN}}$ causes both the transmitter and receiver to achieve synchronization. The SMC transmitter can be disabled and reenabled and $\overline{\text{SMSYN}}$ can be used again to resynchronize the transmitter itself. Section 29.2.4, “Disabling SMCs On-the-Fly,” describes how to safely disable and reenable the SMC. Simply clearing and setting TEN may be insufficient. The receiver can also be resynchronized this way.

29.4.6 Using TSA for Synchronization

The TSA offers an alternative to using $\overline{\text{SMSYN}}$ to internally synchronize the SMC channel. This method is similar, except that the synchronization event is the first time-slot for this

SMC receiver/transmitter after the frame sync indication rather than the falling edge of SMSYN. Chapter 20, “Serial Interface,” describes how to configure time slots. The TSA allows the SMC receiver and transmitter to be enabled simultaneously and synchronized separately; SMSYN does not provide this capability. Figure 29-12 shows synchronization using the TSA.

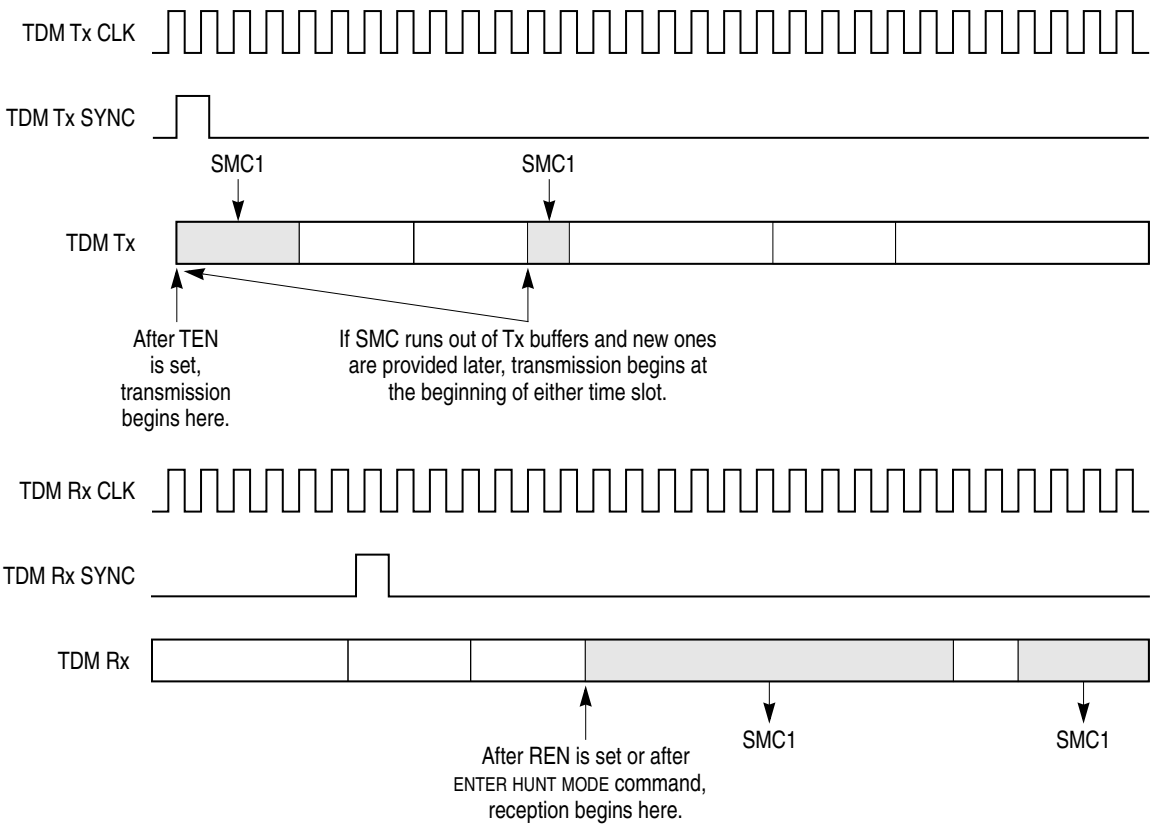


Figure 29-12. Synchronization with the TSA

Once SMCMR[REN] is set, the first time-slot after the frame sync causes the SMC receiver to achieve synchronization. Data is received immediately, but only during defined receive time slots. The receiver continues receiving data during its defined time slots until REN is cleared. If an ENTER HUNT MODE command is issued, the receiver loses synchronization, closes the buffer, and resynchronizes to the first time slot after the frame sync.

Once SMCMR[TEN] is set, the SMC waits for the transmit FIFO to be loaded before trying to achieve synchronization. If only a single time slot in a TDM frame is assigned to the SMC, USMC transmission, as well as reception, is always synchronized to the beginning of that time slot. If multiple time slots in a TDM frame are assigned to the SMC (as shown in Figure 29-12), then synchronization depends on the order of initialization.

When the transmit FIFO is loaded, synchronization and transmission begins depending on the following:

- If a buffer is made ready before the SMC is enabled, the first byte is placed in time slot 1 if CLEN is 8 and to slot 2 if CLEN is greater than 8.
- If a buffer is made ready after its SMC is enabled, the first byte can appear in any time slot associated with this channel.
- If a buffer is closed with BD[L] set, then the next buffer can appear in any time slot associated with this channel.

If the SMC runs out of transmit buffers and a new buffer is provided later, idles are sent in the gap between buffers. Data transmission from the later buffer begins at the start of an SMC time slot, but not necessarily the first time slot after the frame sync. So, to maintain a certain bit alignment beginning with the first time slot, make sure that at least one TxBD is always ready and that underruns do not occur. Otherwise, the SMC transmitter should be disabled and reenabled. Section 29.2.4, “Disabling SMCs On-the-Fly,” describes how to safely disable and reenable the SMC. Simply clearing and setting TEN may not be enough.

29.4.7 SMC Transparent Commands

Table 29-11 describes transmit commands issued to the CPCR.

Table 29-11. SMC Transparent Transmit Commands

Command	Description
STOP TRANSMIT	After hardware or software is reset and the channel is enabled in the SMCMR, the channel is in transmit enable mode and polls the first BD. This command disables transmission of frames on the transmit channel. If the transparent controller receives this command while sending a frame, it stops after the contents of the FIFO are sent (up to 2 characters). The TBPTR is not advanced to the next BD, no new BD is accessed, and no new buffers are sent for this channel. The transmitter sends idles until a RESTART TRANSMIT command is issued.
RESTART TRANSMIT	Starts or resumes transmission from the current TBPTR in the channel TxBD table. When the channel receives this command, it polls the R bit in this BD. The SMC expects this command after a STOP TRANSMIT is issued. The channel is disabled in its mode register or after a transmitter error occurs. In addition, the transmitter awaits resynchronization before transmission continues.
INIT TX PARAMETERS	Initializes transmit parameters in this serial channel to reset state. Use only if the transmitter is disabled. The INIT TX AND RX PARAMETERS command resets transmit and receive parameters.

Figure 29-12 describes receive commands issued to the CPCR.

Table 29-12. SMC Transparent Receive Commands

Command	Description
ENTER HUNT MODE	Forces the SMC to close the current receive BD if it is in use and to use the next BD for subsequent data. If the SMC is not receiving data, the buffer is not closed. Additionally, this command causes the receiver to wait for a resynchronization before reception resumes.

Table 29-12. SMC Transparent Receive Commands (continued)

Command	Description
CLOSE RXBD	Forces the SMC to close the current receive BD if it in use and to use the next BD in the list for subsequent received data. If the SMC is not in the process of receiving data, no action is taken.
INIT RX PARAMETERS	Initializes receive parameters in this serial channel to reset state. Use only if the receiver is disabled. The INIT TX AND RX PARAMETERS command resets receive and transmit parameters.

29.4.8 Handling Errors in the SMC Transparent Controller

The SMC uses BDs and the SMCE to report message transmit and receive errors.

Table 29-13. SMC Transparent Error Conditions

Error	Descriptions
Underrun	The channel stops sending the buffer, closes it, sets UN in the BD, and generates a TXE interrupt if it is enabled. The channel resumes sending after a RESTART TRANSMIT command. Underrun cannot occur between frames.
Overrun	The SMC maintains an internal FIFO for receiving data. If the buffer is in external memory, the CP begins programming the SDMA channel when the first character is received into the FIFO. If a FIFO overrun occurs, the SMC writes the received data character over the previously received character. The previous character and its status bits are lost. Then the channel closes the buffer, sets OV in the BD, and generates the RX interrupt if it is enabled. Reception continues as normal.

29.4.9 SMC Transparent Receive BD (RxBD)

Using BDs, the CP 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:

- An overrun error occurs.
- A full receive buffer is detected.
- The ENTER HUNT MODE command is issued.

Figure 29-13 shows the SMC transparent RxBD format.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**E**	—	**W**	**I**	—		**CM**	—							OV	—
Offset + 2	Data Length															
Offset + 4	**Rx Buffer Pointer**															
Offset + 6																

Figure 29-13. SMC Transparent Receive BD (RxBD)

Table 29-14 describes SMC transparent RxBD fields.

Table 29-14. SMC Transparent RxBD Field Descriptions

Bits	Name	Description
0	**E**	Empty. 0 The buffer is full or reception was aborted due to an error. The core can read or write any fields of this RxBD. The CP does not use this BD while E = 0. 1 The buffer is empty or is receiving data. The CP owns this RxBD and its buffer. Once E is set, the core should not write any fields of this RxBD.
1	—	Reserved, should be cleared.
2	**W**	Wrap (last BD in RxBD table). 0 Not the last BD in the table. 1 Last BD in the table. After this buffer is used, the CP receives incoming data into the first BD that RBASE points to. The number of RxBDs is determined only by the W bit.
3	**I**	Interrupt. 0 No interrupt is generated after this buffer is filled. 1 SMCE[RX] is set when the CP completely fills this buffer indicating that the core must process the buffer. The RX bit can cause an interrupt if it is enabled.
4–5	—	Reserved, should be cleared.
6	**CM**	Continuous mode. 0 Normal operation. 1 The CP does not clear E after this BD is closed, allowing the buffer to be overwritten when the CP next accesses this BD. However, E is cleared if an error occurs during reception, regardless of how CM is set.
7–13	—	Reserved, should be cleared.
14	OV	Overrun. Set when a receiver overrun occurs during reception. The CP writes OV after the received data is placed into the buffer.
15	—	Reserved, should be cleared.

Data length and buffer pointer fields are described in Section 21.3, “SCC Buffer Descriptors (BDs).”

29.4.10 SMC Transparent Transmit BD (TxBD)

Data is sent to the CPM for transmission on an SMC channel by arranging it in buffers referenced by the channel TxBD table. The CP uses BDs to confirm transmission or indicate error conditions so the processor knows buffers have been serviced.

Figure 29-14 shows the SMC transparent TxBD format.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**R**	—	**W**	**I**	**L**	—	**CM**	—							UN	—
Offset + 2	**Data Length**															
Offset + 4	**Tx Buffer Pointer**															
Offset + 6																

Figure 29-14. SMC Transparent Transmit BD (TxBD)

Table 29-15 describes SMC transparent TxBD fields.

Table 29-15. SMC Transparent TxBD Field Descriptions

Bits	Name	Description
0	**R**	Ready. 0 The buffer is not ready for transmission. The BD and buffer can be updated. The CP clears R after the buffer is sent or after an error occurs. 1 The user-prepared buffer is not sent or is being sent. BD fields must not be updated if R is set.
1	—	Reserved, should be cleared.
2	**W**	Wrap (final BD in table). 0 Not the last BD in the table. 1 Last BD in the table. After this buffer is used, the CP receives incoming data into the first BD that TBASE points to. The number of TxBDs in this table is programmable and determined by the W bit.
3	**I**	Interrupt. 0 No interrupt is generated after this buffer is serviced unless an error occurs. 1 SMCE[TX] or SMCE[TXE] are set when the buffer is serviced. They can cause interrupts if they are enabled. Note that this bit does not mask SMCE[TXE].
4	**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 sent immediately after 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 sent, the transmitter requires synchronization before the next buffer is sent.
5	—	Reserved, should be cleared.
6	**CM**	Continuous mode. 0 Normal operation. 1 The CP does not clear R after this BD is closed, allowing the buffer to be automatically resent when the CP accesses this BD again. However, the R bit is cleared if an error occurs during transmission, regardless of how CM is set.
7–13	—	Reserved, should be cleared.
14	UM	Underrun. Set when the SMC encounters a transmitter underrun condition while sending the buffer.
15	—	Reserved, should be cleared.

Data length represents the number of octets the CP should transmit from this buffer. It is never modified by the CP. The data length can be even or odd, but if the number of bits in the transparent character is greater than 8, the data length should be even. For example, to transmit three transparent 8-bit characters, the data length field should be initialized to 3.

However, to transmit three transparent 9-bit characters, the data length field should be initialized to 6 because the three 9-bit characters occupy three half words in memory.

The buffer pointer points to the first byte of the buffer. They can be even or odd, unless character length is greater than 8 bits, in which case the transmit buffer pointer 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.

29.4.11 SMC Transparent Event Register (SMCE)/ Mask Register (SMCM)

The SMC event register (SMCE) generates interrupts and reports events recognized by the SMC channel. When an event is recognized, the SMC sets the corresponding SMCE bit. Interrupts are masked in the SMCM, which has the same format as the SMCE. SMCE bits are cleared by writing ones; writing zeros has no effect. Unmasked bits must be cleared before the CP clears the internal interrupt request.

Figure 29-15 shows the SMCE/SMCM register format.

Bit	0	1	2	3	4	5	6	7
Field	—			TXE	—	BSY	TX	RX
Reset	0							
R/W	R/W							
Address	0xA86 (SMCE1), 0xA96 (SMCE2)/ 0xA8A (SMCM1), 0xA9A (SMCM2)							

Figure 29-15. SMC Transparent Event Register (SMCE)/Mask Register (SMCM)

Table 29-16 describes SMCE/SMCM fields.

Table 29-16. SMCE/SMCM Field Descriptions

Bits	Name	Description
0–2	—	Reserved, should be cleared.
3	TXE	Tx error. Set when an underrun error occurs on the transmitter channel.
4	—	Reserved, should be cleared.
5	BSY	Busy condition. Set when a character is received and discarded due to a lack of buffers. Reception begins after a new buffer is provided, without waiting for resynchronization. To resynchronize after error recovery, issue an ENTER HUNT MODE command.
6	TX	Tx buffer. Set after a buffer is sent. If the L bit of the TxBD is set, TX is set when the last character starts being sent. A one character-time delay is required to ensure that data is completely sent over the transmit pin. If the L bit of the TxBD is cleared, TX is set when the last character is written to the transmit FIFO. A two character-time delay is required to ensure that data is completely sent.
7	RX	Rx buffer. Set when a buffer is received (after the last character is written) on the SMC channel and its associated RxBD is closed.

29.4.12 SMC Transparent NMSI Programming Example

The following example initializes the SMC1 transparent channel over its own set of pins. The CLK3 pin supplies the transmit and receive clocks; the $\overline{\text{SMSYN}}x$ pin is used for synchronization. The SMC UART example shows baud-rate generator configuration.

1. Configure the port B pins to enable SMTXD1, SMRXD1, and $\overline{\text{SMSYN1}}$. Set PBPAR[23– 25] and clear PBDIR[23– 25] and PBODR[23– 25].
2. Configure the port A pins to enable CLK3. Set PAPAR[5] and clear PADIR[5]. The other pin functions are the timers or the TSA. These alternate functions cannot be used on this pin.
3. Connect CLK3 to SMC1 using the SI. Clear SIMODE[SMC1] and set SIMODE[SMC1CS] to 0b110.
4. Write RBASE and TBASE in the SMC parameter RAM to point to the RxBD and TxBD in the dual-port RAM. Assuming one RxBD at the beginning of the dual-port RAM followed by one TxBD, write RBASE with 0x0000 and TBASE with 0x0008.
5. Program the CPCR to execute the INIT RX AND TX PARAMETERS command. Write 0x0091 to the CPCR.
6. Write 0x0001 to the SDCR to initialize the SDCR.
7. Write RFCR and TFCR with 0x10 for normal operation.
8. Write MRBLR with the maximum bytes per receive buffer. Assuming 16 bytes, MRBLR = 0x0010.
9. Initialize the RxBD assuming the buffer is at 0x0000_1000 in main memory. Write 0xB000 to RxBD[Status and Control], 0x0000 to RxBD[Data Length] (optional), and 0x0000_1000 to RxBD[Buffer Pointer].
10. Initialize the TxBD assuming the Tx buffer is at 0x0000_2000 in main memory and contains five 8-bit characters. Write 0xB000 to TxBD[Status and Control], 0x0005 to TxBD[Data Length], and 0x0000_2000 to TxBD[Buffer Pointer].
11. Write 0xFF to SMCE to clear any previous events.
12. Write 0x13 to SMCM to enable all possible SMC interrupts.
13. Write 0x0000_0010 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). The transmitter and receiver are not enabled yet.
15. Write 0x3833 to the SMCMR to enable the SMC transmitter and receiver. This additional write ensures that TEN and REN are enabled last.

After 5 bytes are sent, the TxBD is closed; after 16 bytes are received the receive buffer is closed. Any data received after 16 bytes causes a busy (out-of-buffers) condition since only one RxBD is prepared.

29.4.13 SMC Transparent TSA Programming Example

The following is an example initialization sequence for the SMC1 transparent channel over the TSA. It is assumed that the TSA and the TDM pins have been set up to route time-slot

data to the SMC transmitter and receiver. Chapter 20, “Serial Interface,” has examples for configuring the TSA which provides transmit and receive clocks and synchronization signals internally.

1. Write RBASE and TBASE in the SMC parameter RAM to point to the RxBD and TxBD in the dual-port RAM. Assuming one RxBD at the beginning of the dual-port RAM followed by one TxBD, write RBASE with 0x0000 and TBASE with 0x0008.
2. Program CPCR to execute the INIT TX AND RX PARAMETERS command. Write 0x0091 to the CPCR.
3. Initialize the SDCR to 0x0001.
4. Write RFCR and TFCR with 0x10 for normal operation.
5. Write MRBLR with the maximum number of bytes per receive buffer. Assume 16 bytes, so MRBLR = 0x0010.
6. Initialize the RxBD and assume the Rx buffer is at 0x0000_1000 in main memory. Write 0xB000 to RxBD[Status and Control], 0x0000 to RxBD[Data Length] (optional), and 0x0000_1000 to RxBD[Buffer Pointer].
7. Initialize the TxBD and assume the Tx buffer is at 0x0000_2000 in main memory and contains five 8-bit characters. Write 0xB000 to TxBD[Status and Control], 0x0005 to TxBD[Data Length], and 0x0000_2000 to TxBD[Buffer Pointer].
8. Write 0xFF to SMCE to clear any previous events.
9. Write 0x13 to SMCM to enable all possible SMC interrupts.
10. Write 0x0000_0010 to the CIMR so SMC1 can generate a system interrupt. Initialize CICR.
11. Set SMCMR to 0x3830 for 8-bit characters, unreversed data, and normal operation (not loopback). The transmitter and receiver are not enabled yet.
12. Write 0x3833 to SMCMR to enable the SMC transmitter and receiver. This additional write ensures that TEN and REN are enabled last.

29.5 SMC in GCI Mode

A single SMC can control both the C/I and monitor channels of a GCI frame. When using the SCIT configuration of GCI, one SMC can handle SCIT channel 0 and the other can handle SCIT channel 1. The main features of the SMC in GCI mode are as follows:

- Each SMC channel can support both the C/I and monitor channels of the GCI (IOM-2) in ISDN applications
- Two SMCs support both sets of C/I 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 TSA must be configured to route the monitor and C/I channels to the preferred SMC. Chapter 20, “Serial Interface,” describes how to program this configuration. GCI mode is selected by programming SMCMR[SM] to 0b00. Section 29.2.1, “SMC Mode Registers (SMCMRn),” describes other protocol-specific SMCMR bits.

29.5.1 SMC GCI Parameter RAM

The SMC GCI parameter RAM area begins at the same offset from each SMC base. The parameter RAM differs from that for UART and transparent mode. In GCI mode, parameter RAM contains both the BDs and their buffers. Compare Table 29-17 with Table 29-2 to see the differences. In GCI mode the SMC has no protocol-specific parameter RAM.

Table 29-17. SMC GCI Parameter RAM Memory Map

Offset [1]	Name	Width	Description
0x00	**M_RxBD**	Hword	Monitor channel RxBD. See Section 29.5.5, “SMC GCI Monitor Channel RxBD.”
0x02	**M_TxBD**	Hword	Monitor channel TxBD. See Section 29.5.6, “SMC GCI Monitor Channel TxBD.”
0x04	**CI_RxBD**	Hword	C/I channel RxBD. See Section 29.5.7, “SMC GCI C/I Channel RxBD.”
0x06	**CI_TxBD**	Hword	C/I channel TxBD. See Section 29.5.8, “SMC GCI C/I Channel TxBD.”
0x08	RSTATE [2]	Word	Rx/ Tx Internal State
0x0C	M_RxD [2]	Hword	Monitor Rx Data
0x0E	M_TxD [2]	Hword	Monitor Tx Data
0x10	CI_RxD [2]	Hword	C/I Rx Data
0x12	CI_TxD [2]	Hword	C/I Tx Data

[1] SMC base = IMMR + 0x3E80 (SMC1), 0x3F80 (SMC2).

[2] RSTATE, M_RxD, M_TxD, CI_RxD, and CI_TxD do not need to be accessed by the user in normal operation, and are reserved for CP use only.

29.5.2 Handling the GCI Monitor Channel

The following sections describe how the GCI monitor channel is handled.

29.5.2.1 SMC GCI Monitor Channel Transmission Process

Monitor channel 0 is used to exchange data with an OSI layer 1 device (reading and writing internal registers and transferring of the S and Q bits). In SCIT configuration, monitor channel 1 is used for programming and controlling voice/data modules such as CODECs. The core writes the byte into the TxBD. The SMC sends the data on the monitor channel and handles the A and E control bits according to the GCI monitor channel protocol. The

TIMEOUT command resolves deadlocks when errors in the A and E bit states occur on the data line.

29.5.2.1.1 SMC GCI Monitor Channel Reception Process

The SMC receives data and handles the A and E control bits according to the GCI monitor channel protocol. When the CP stores a received data byte in the SMC RxBD, a maskable interrupt is generated. A TRANSMIT ABORT REQUEST command causes the MPC855T to send an abort request on the E bit.

29.5.3 Handling the GCI C/I Channel

The C/I channel is used to control the OSI layer 1 device. The OSI layer 2 device in the TE sends commands and receives indication to or from the upstream layer 1 device through C/I channel 0. In the SCIT configuration, C/I channel 1 is used to convey real-time status information between the layer 2 device and nonlayer 1 peripheral devices (CODECs).

29.5.3.1 SMC GCI C/I Channel Transmission Process

The core writes the data byte into the C/I TxBD and the SMC transmits the data continuously on the C/I channel to the physical layer device.

29.5.3.2 SMC GCI C/I Channel Reception Process

The SMC receiver continuously monitors the C/I channel. 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 called the double last-look method. The CP stores the received data byte in the C/I RxBD and a maskable interrupt is generated. If the SMC is configured to support SCIT channel 1, the double last-look method is not used.

29.5.4 SMC GCI Commands

The commands in Table 29-18 are issued to the CPCR.

Table 29-18. SMC GCI Commands

Command	Description
INIT TX AND RX PARAMETERS	Initializes transmit and receive parameters in the parameter RAM to their reset state.
TRANSMIT ABORT REQUEST	This receiver command can be issued when the MPC855T implements the monitor channel protocol. When it is issued, the MPC855T sends an abort request on the A bit.
TIMEOUT	This transmitter command can be issued when the MPC855T implements the monitor channel protocol. It is usually issued because the device is not responding or A bit errors are detected. The MPC855T sends an abort request on the E bit at the time this command is issued.

29.5.5 SMC GCI Monitor Channel RxBD

The GCI monitor channel RxBD, shown in Figure 29-16, is used by the CP to report on the monitor channel receive byte. The RxBD itself receives the monitor data.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**E**	L	ER	MS	—				Data							

Figure 29-16. SMC GCI Monitor Channel RxBD

Table 29-19 describes SMC monitor channel RxBD fields.

Table 29-19. SMC Monitor Channel RxBD Field Descriptions

Bits	Name	Description
0	**E**	Empty. 0 The CP clears E when the byte associated with this BD is available to the core. 1 The core sets E when the byte associated with this BD has been read.
1	L	Last (EOM). Set when the EOM indication is received on the E bit. Note that when this bit is set, the data byte is invalid.
2	ER	Error condition. Set when an error occurs on the monitor channel protocol. The error condition indicates that a new byte was sent before the SMC acknowledged the previous byte.
3	MS	Data mismatch. Set when two different consecutive bytes are received; cleared when the last two consecutive bytes match. The SMC waits for the reception of two identical consecutive bytes before writing new data to the RxBD.
4–7	—	Reserved, should be cleared.
8–15	Data	Data field. Contains the monitor channel data byte that the SMC received.

29.5.6 SMC GCI Monitor Channel TxBD

The CP uses the GCI monitor channel TxBD, shown in Figure 29-17, to report on the monitor channel transmit byte. The TxBD itself contains the monitor data to be sent.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**R**	**L**	AR	—					**Data**							

Figure 29-17. SMC GCI Monitor Channel TxBD

Table 29-20 describes SMC monitor channel TxBD fields.

Table 29-20. SMC Monitor Channel TxBD Field Descriptions

Bits	Name	Description
0	**R**	Ready. 0 Cleared by the CP after transmission. The TxBD is now available to the core. 1 Set by the core when the data byte associated with this BD is ready for transmission.
1	**L**	Last (EOM). When L = 1, the SMC first transmits the buffer data and then transmits the EOM indication on the E bit.

Table 29-20. SMC Monitor Channel TxBD Field Descriptions (continued)

Bits	Name	Description
2	AR	Abort request. Set by the SMC when an abort request is received on the A bit. The transmitter sends the EOM on the E bit after receiving an abort request.
3–7	—	Reserved, should be cleared.
8–15	**Data**	Data field. Contains the data to be sent by the SMC on the monitor channel.

29.5.7 SMC GCI C/I Channel RxBD

The GCI C/I channel RxBD, shown in Figure 29-18, is used by the CP to report on the C/I channel receive byte. The RxBD itself receives the C/I data.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**E**	—							C/I Data						—	

Figure 29-18. SMC C/I Channel RxBD

Table 29-21 describes SMC C/I channel RxBD fields

Table 29-21. SMC C/I Channel RxBD Field Descriptions

Bits	Name	Description
0	**E**	Empty. 0 Cleared by the CP to indicate that the byte associated with this BD is available to the core. 1 The core sets E to indicate that the byte associated with this BD has been read. Note that additional data received is discarded until E bit is set.
1–7	—	Reserved, should be cleared.
8–13	C/I Data	Command/indication data bits. For C/I channel 0, bits 10–13 contain the 4-bit data field and bits 8–9 are always written with zeros. For C/I channel 1, bits 8–13 contain the 6-bit data field.
14–15	—	Reserved, should be cleared.

29.5.8 SMC GCI C/I Channel TxBD

The GCI C/I channel TxBD, shown in Figure 29-19, is used by the CP to report on the C/I channel transmit byte. The TxBD itself contains the C/I data to be sent.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**R**	—							**C/I Data**						—	

Figure 29-19. SMC C/I Channel TxBD

Table 29-22 describes SMC C/I channel TxBD fields.

Table 29-22. SMC C/I Channel TxBD Field Descriptions

Bits	Name	Description
0	**R**	Ready. 0 Cleared by the CP after transmission to indicate that the BD is available to the core. 1 Set by the core when data associated with this BD is ready for transmission.
1–7	—	Reserved, should be cleared.
8–13	**C/I Data**	Command/indication data bits. For C/I channel 0, bits 10–13 hold the 4-bit data field (bits 8 and 9 should be written with zeros). For C/I channel 1, bits 8–13 contain the 6-bit data field.
14–15	—	Reserved, should be cleared.

29.5.9 SMC GCI Event Register (SMCE)/ Mask Register (SMCM)

The SMCE generates interrupts and reports events recognized by the SMC channel. When an event is recognized, the SMC sets its corresponding SMCE bit. SMCE bits are cleared by writing ones; writing zeros has no effect. SMCM has the same bit format as SMCE. Setting an SMCM bit enables, and clearing an SMCM bit disables, the corresponding interrupt Unmasked bits must be cleared before the CP clears the internal interrupt request to the CP interrupt controller (CPIC). Figure 29-20 shows the SMCE/SMCM register format.

Bit	0	1	2	3	4	5	6	7
Field	—				CTXB	CRXB	MTXB	MRXB
Reset	0000_0000							
R/W	R/W							
Address	0xA86 (SMCE1), 0xA96 (SMCE2)/ 0xA8A (SMCM1), 0xA9A (SMCM2)							

Figure 29-20. SMC GCI Event Register (SMCE)/Mask Register (SMCM)

Table 29-23 describes SMCE/SMCM fields.

Table 29-23. SMCE/SMCM Field Descriptions

Bits	Name	Description
0–3	—	Reserved, should be cleared.
4	CTXB	C/I channel buffer transmitted. Set when the C/I transmit buffer becomes empty.
5	CRXB	C/I channel buffer received. Set when the C/I receive buffer becomes full.
6	MTXB	Monitor channel buffer transmitted. Set when the monitor transmit buffer becomes empty.
7	MRXB	Monitor channel buffer received. Set when the monitor receive buffer becomes full.

Chapter 30
Serial Peripheral Interface (SPI)

The serial peripheral interface (SPI) allows the MPC855T to exchange data between other MPC855T chips, the MC68360, the MC68302, the M68HC11 and M68HC05 microcontroller families, and peripheral devices such as EEPROMs, real-time clocks, A/D converters, and ISDN devices.

The SPI 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 sent and received simultaneously.

Because the SPI receiver and transmitter are double-buffered, as shown in Figure 30-1, the effective FIFO size (latency) is 2 characters. The SPI's msb is shifted out first. When the SPI is disabled in the SPI mode register (SPMODE[EN] = 0), it consumes little power.

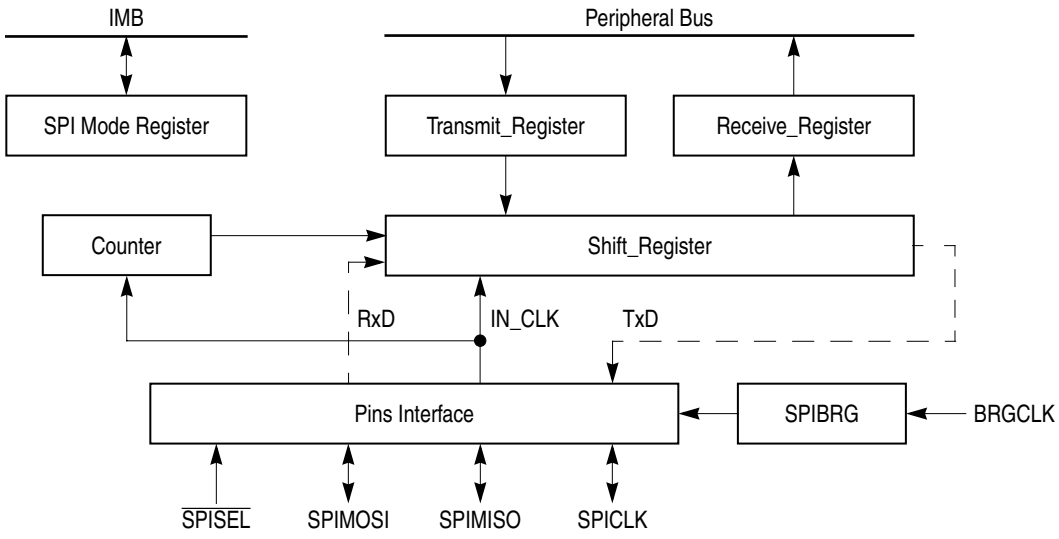


Figure 30-1. SPI Block Diagram

30.1 Features

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

- Four-signal 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 of a peripheral
- Supports maximum clock rates of 6.25 MHz in master mode and12.5 MHz in slave mode, assuming a 25-MHz system clock
- Independent programmable baud rate generator
- Programmable clock phase and polarity
- Open-drain outputs support multimaster configuration
- Local loopback capability for testing

30.2 SPI Clocking and Signal Functions

The SPI can be configured as a slave or as a master in single- or multiple-master environments. The master SPI generates the transfer clock SPICLK using the SPI baud rate generator (BRG). The SPI BRG takes its input from BRGCLK, which is generated in the MPC855T clock synthesizer.

SPICLK is a gated clock, active only during data transfers. Therefore, SPI clock rates can be very high, up to BRGCLK/4 in master mode or BRGCLK/2 in slave mode. Note, however, that this high clock rate can be supported only over the period of a single character, if messages consist of multiple back-to-back characters, operation becomes limited by CPM performance, and thus the clock rate should be adjusted down accordingly. CPM bandwidth required by the SPI channel should be calculated as the maximum rate that back-to-back characters must be transmitted and received. Four combinations of SPICLK phase and polarity can be configured with SPMODE[CI, CP]. SPI signals can also be configured as open-drain to support a multimaster configuration in which a shared SPI signal is driven by the MPC855T or an external SPI device.

The SPI master-in slave-out SPIMISO signal acts as an input for master devices and as an output for slave devices. Conversely, the master-out slave-in SPIMOSI signal is an output for master devices and an input for slave devices. The dual functionality of these signals allows the SPIs in a multimaster environment to communicate with one another using a common hardware configuration.

- When the SPI is a master, SPICLK is the clock output signal that shifts received data in from SPIMISO and transmitted data out to SPIMOSI. SPI masters must output a slave select signal to enable SPI slave devices by using a separate general-purpose I/O signal. Assertion of an SPI's $\overline{\text{SPISEL}}$, while it is in master mode, causes an error.
- When the SPI is a slave, SPICLK is the clock input that shifts received data in from SPIMOSI and transmitted data out through SPIMISO. $\overline{\text{SPISEL}}$ is the enable input to the SPI slave.
- In a multimaster environment, $\overline{\text{SPISEL}}$ (always an input) is also used to detect when more than one master is operating, which is an error condition.

As described in Chapter 33, "Parallel I/O Ports," SPIMISO, SPIMOSI, SPICLK, and $\overline{\text{SPISEL}}$ are multiplexed with port B[28–31] signals, respectively. They are configured as SPI signals through the port B signal assignment register (PBPAR) and the Port B data direction register (PBDIR), specifically by setting PBPAR[DD*n*] and PBDIR[DR*n*].

30.3 Configuring the SPI Controller

The SPI can be programmed to work in a single- or multiple-master environment. This section describes SPI master and slave operation in a single-master configuration and then discusses the multi-master environment.

SPI transmission and reception will always be enabled simultaneously. If the transmit or receive function is not needed, the user can point the associated channel of a non-ready TxBD or RxBD, and simply ignore the resultant Tx underrun or Rx busy errors.

30.3.1 The SPI as a Master Device

In master mode, the SPI sends a message to the slave peripheral, which sends back a simultaneous reply. A single master MPC855T with multiple slaves can use general-purpose parallel I/O signals to selectively enable slaves, as shown in Figure 30-2. To eliminate the multimaster error in a single-master environment, the master's $\overline{\text{SPISEL}}$ input can be forced inactive by selecting port B[31] for general-purpose I/O (PBPAR[DD31] = 0).

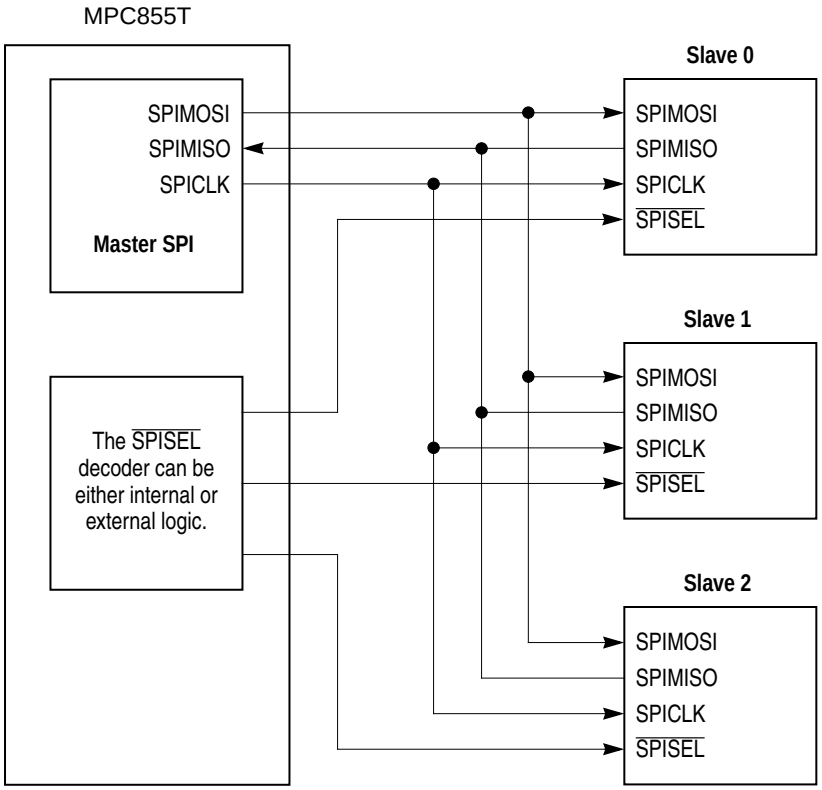


Figure 30-2. Single-Master/Multi-Slave Configuration

To start exchanging data, the core writes the data to be sent into a buffer, configures a TxBD with TxBD[R] set, and configures one or more RxBDs. The core then sets SPCOM[STR] in the SPI command register to start sending data, which starts once the SDMA channel loads the Tx FIFO with data.

The SPI then generates programmable clock pulses on SPICLK for each character and simultaneously shifts Tx data out on SPIMOSI and Rx data in on SPIMISO. Received data is written into a Rx buffer using the next available RxBD. The SPI keeps sending and receiving characters until the whole buffer is sent or an error occurs. The CPM then clears TxBD[R] and RxBD[E] and issues a maskable interrupt to the CPM interrupt controller (CPIC).

When multiple TxBDs are ready, TxBD[L] determines whether the SPI keeps transmitting without SPCOM[STR] being set again. If the current TxBD[L] is cleared, the next TxBD is processed after data from the current buffer is sent. Typically there is no delay on SPIMOSI between buffers. If the current TxBD[L] is set, sending stops after the current buffer is sent. In addition, the RxBD is closed after transmission stops, even if the Rx buffer is not full; therefore, Rx buffers need not be the same length as Tx buffers.

30.3.2 The SPI as a Slave Device

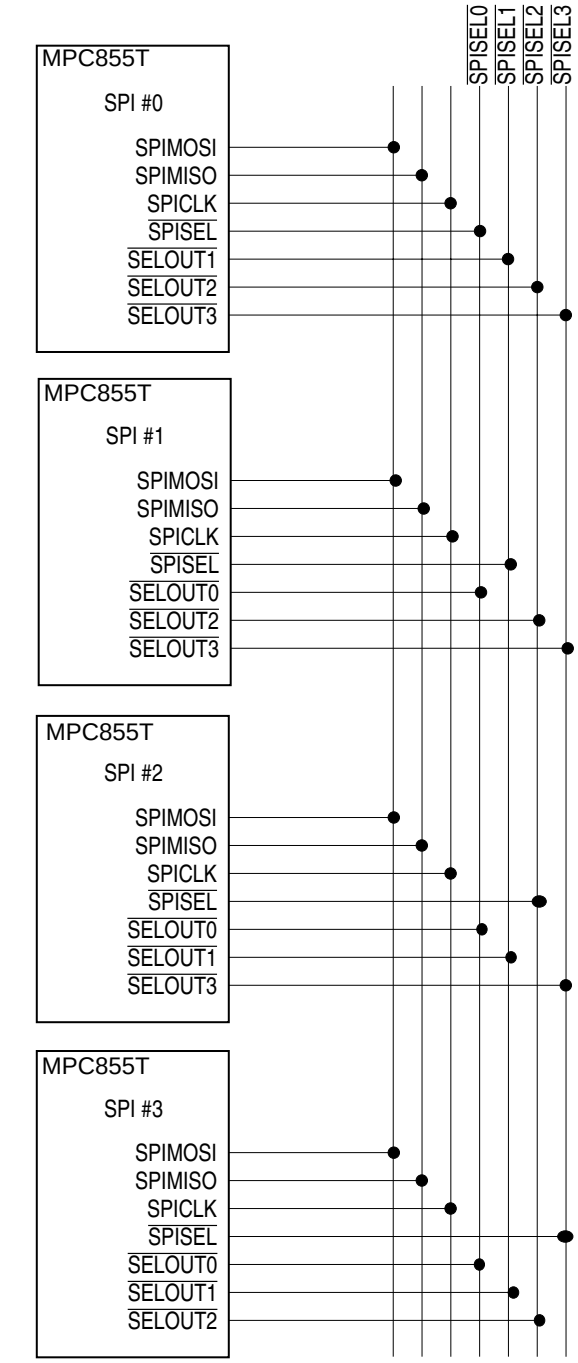
In slave mode, the SPI receives messages from an SPI master and sends a simultaneous reply. The slave's $\overline{\text{SPISEL}}$ must be asserted before Rx clocks are recognized; once $\overline{\text{SPISEL}}$ is asserted, SPICLK becomes an input from the master to the slave. SPICLK can be any frequency from DC to BRGCLK/2 (12.5 MHz for a 25-MHz system).

To prepare for data transfers, the slave's core writes data to be sent into a buffer, configures a TxBD with TxBD[R] set, and configures one or more RxBDs. The core then sets SPCOM[STR] to activate the SPI. Once $\overline{\text{SPISEL}}$ is asserted, the slave shifts data out from SPIMISO and in through SPIMOSI. A maskable interrupt is issued when a full buffer finishes receiving and sending or after an error. The SPI uses successive RxBDs in the table to continue reception until it runs out of Rx buffers or $\overline{\text{SPISEL}}$ is negated.

Transmission continues until no more data is available or $\overline{\text{SPISEL}}$ is negated. If it is negated before all data is sent, it stops but the TxBD stays open. Transmission continues once $\overline{\text{SPISEL}}$ is reasserted and SPICLK begins toggling. After the characters in the buffer are sent, the SPI sends ones as long as $\overline{\text{SPISEL}}$ remains asserted.

30.3.3 The SPI in Multi-master Operation

The SPI can operate in a multimaster environment in which SPI devices are connected to the same bus. In this configuration, the SPIMOSI, SPIMISO, and SPICLK signals of all SPIs are shared; the $\overline{\text{SPISEL}}$ inputs are connected separately, as shown in Figure 30-3. Only one SPI device can act as master at a time—all others must be slaves. When an SPI is configured as a master and its $\overline{\text{SPISEL}}$ input is asserted, a multimaster error occurs because more than one SPI device is a bus master. The SPI sets SPIE[MME] in the SPI event register and a maskable interrupt is issued to the core. It also disables SPI operation and the output drivers of SPI signals. The core must clear SPMODE[EN] before the SPI is used again. After correcting the problems, clear SPIE[MME] and reenable the SPI.



Notes:
- All signals are open-drain
- For a multi-master system with more than two masters, SPISEL and SPIE[MME] will not detect all possible conflicts
- It is the responsibility of software to arbitrate for the SPI bus (with token passing, for example)
- SELOUT*x* signals are implemented in software with general-purpose I/O signals

Figure 30-3. Multimaster Configuration

30.4 SPI Registers

The following sections describe the registers used in configuring and operating the SPI.

30.4.1 SPI Mode Register (SPMODE)

The SPI mode register (SPMODE), shown in Figure 30-4, controls both the SPI operation mode and clock source.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—	LOOP	CI	CP	DIV16	REV	M/S	EN	LEN				PM			
Reset	0000_00						—	0_0000_0000								
R/W	R/W															
Addr	0xAA0															

Figure 30-4. SPI Mode Register (SPMODE)

This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Table 30-1 describes the SPMODE fields.

Table 30-1. SPMODE Field Descriptions

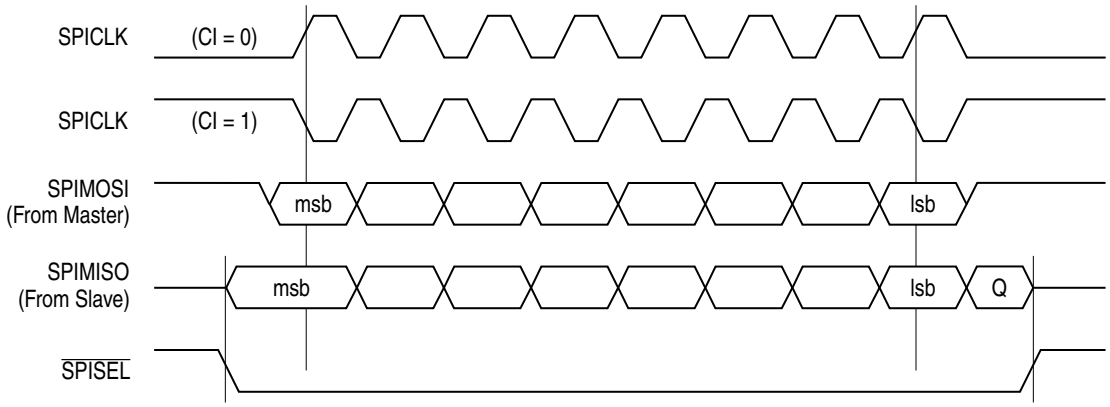
Bits	Name	Description
0	—	Reserved, should be cleared.
1	LOOP	Loop mode. Enables local loopback operation. 0 Normal operation. 1 Loopback mode. The transmitter output is internally connected to the receiver input. The receiver and transmitter operate normally, except that received data is ignored.
2	CI	Clock invert. Inverts SPI clock polarity. See Figure 30-5 and Figure 30-6. 0 The inactive state of SPICLK is low. 1 The inactive state of SPICLK is high.
3	CP	Clock phase. Selects the transfer format. See Figure 30-5 and Figure 30-6. 0 SPICLK starts toggling at the middle of the data transfer. 1 SPICLK starts toggling at the beginning of the data transfer.
4	DIV16	Divide by 16. Selects the clock source for the SPI baud rate generator when configured as an SPI master. In slave mode, SPICLK is the clock source. 0 BRGCLK is the input to the SPI BRG. 1 BRGCLK/16 is the input to the SPI BRG.
5	REV	Reverse data. Determines the receive and transmit character bit order. 0 Reverse data—lsb of the character sent and received first. 1 Normal operation—msb of the character sent and received first.
6	M/S	Master/slave. Selects master or slave mode. 0 The SPI is a slave. 1 The SPI is a master. Note that master/slave mode is undefined at reset.

Table 30-1. SPMODE Field Descriptions (continued)

Bits	Name	Description
7	EN	Enable SPI. Do not change other SPMODE bits when EN is set. 0 The SPI is disabled. The SPI is in a reset state and consumes minimal power. The SPI BRG is not functioning and the input clock is disabled. 1 The SPI is enabled.
8–11	LEN	Character length in bits per character. Must be between 0011 (4 bits) and 1111 (16 bits). A value less than 4 causes erratic behavior. If the value is not greater than a byte, every byte in memory holds LEN valid bits. If the value is greater than a byte, every half-word holds LEN valid bits. See Section 30.4.1.2, “SPI Examples with Different SPMODE[LEN] Values.”
12–15	PM	Prescale modulus select. Specifies the divide ratio of the prescale divider in the SPI clock generator. BRGCLK is divided by 4 * ([PM0–PM3] + 1), a range from 4 to 64. The clock has a 50% duty cycle.

30.4.1.1 SPI Transfers with Different Clocking Modes

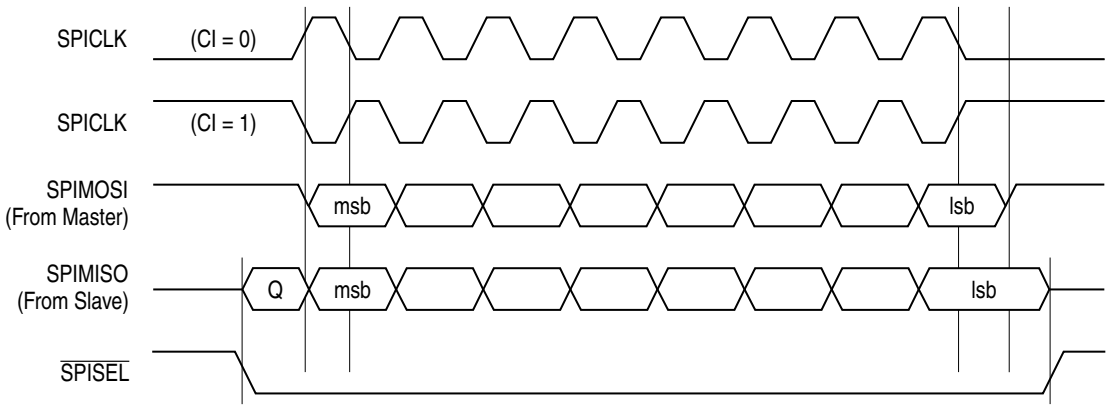
Figure 30-5 shows the SPI transfer format in which SPICLK starts toggling in the middle of the transfer (SPMODE[CP] = 0).



NOTE: Q = Undefined Signal.

Figure 30-5. SPI Transfer Format with SPMODE[CP] = 0

Figure 30-6 shows the SPI transfer format in which SPICLK starts toggling at the beginning of the transfer (SPMODE[CP] = 1).



NOTE: Q = Undefined Signal.

Figure 30-6. SPI Transfer Format with SPMODE[CP] = 1

30.4.1.2 SPI Examples with Different SPMODE[LEN] Values

The examples below show how SPMODE[LEN] is used to determine character length. To help map the process, the conventions shown in Table 30-2 are used in the examples.

Table 30-2. Example Conventions

Convention	Description
g–v	Binary symbols
x	Deleted bit
__ [1]	Original byte boundary
_ [1]	Original 4-bit boundary.

[1] Both __ and _ are used to aid readability.

For all examples below, assume the memory contains the following binary image:

```
              msb         ghij_klmn__opqr_stuv              lsb
```

Example 1

```
with LEN=4 (data size=5), the following data is selected:
              msb         xxxj_klmn__xxxr_stuv              lsb

with REV=0, the order of the string appearing on the line is:
              first       nmlk_j__vuts_r                    last

with REV=1, the order of the string appearing on the line is:
              first       j_klmn__r_stuv                    last
```

Example 2

```
with LEN=7 (data size=8), the following data is selected:
              msb         ghij_klmn__opqr_stuv              lsb
```

```
with REV=0, the string transmitted is:
              first       nmlk_jihg__vuts_rqpo              last

with REV=1, the string is transmitted:
              first       ghij_klmn__opqr_stuv              last
```

Example 3

```
with LEN=0xC (data size=13), the following data is selected:
              msb         ghij_klmn__xxxr_stuv              lsb

with REV=0, the string transmitted:
              first       nmlk_jihg__vuts_r                 last

with REV=1, the string is transmitted:
              first       ghij_klmn__r_stuv                 last
```

30.4.2 SPI Event/Mask Registers (SPIE/SPIM)

The SPI event register (SPIE) generates interrupts and reports events recognized by the SPI. When an event is recognized, the SPI sets the corresponding SPIE bit. Clear SPIE bits by writing a 1—writing 0 has no effect. Setting a bit in the SPI mask register (SPIM) enables and clearing a bit masks the corresponding interrupt. Unmasked SPIE bits must be cleared before the CPM clears internal interrupt requests. Figure 30-7 shows both registers.

Bit	0	1	2	3	4	5	6	7
Field	—		MME	TXE	—	BSY	TXB	RXB
Reset	0							
R/W	R/W							
Addr	0xAA6 (SPIE); 0xAAA (SPIM)							

Figure 30-7. SPI Event/Mask Registers (SPIE/SPIM)

These registers are affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Table 30-3 describes the SPIE/SPIM fields.

Table 30-3. SPIE/SPIM Field Descriptions

Bits	Name	Description
0–1	—	Reserved, should be cleared.
2	MME	Multimaster error. Set when $\overline{\text{SPISEL}}$ is asserted externally while the SPI is in master mode.
3	TXE	Tx error. Set when an error occurs during transmission.
4	—	Reserved, should be cleared.
5	BSY	Busy. Set after the first character is received but discarded because no Rx buffer is available.

Table 30-3. SPIE/SPIM Field Descriptions (continued)

Bits	Name	Description
6	TXB	Tx buffer. Set when the Tx data of the last character in the buffer is written to the Tx FIFO. Wait two character times to be sure data is completely sent over the transmit signal.
7	RXB	Rx buffer. Set after the last character is written to the Rx buffer and the BD is closed.

30.4.3 SPI Command Register (SPCOM)

The SPI command register (SPCOM), shown in Figure 30-8, is used to start SPI operation.

Bit	0	1	2	3	4	5	6	7
Field	STR	—						
Reset	0	0						
R/W	R/W							
Addr	0xAAD							

Figure 30-8. SPI Command Register (SPCOM)

This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Table 30-4 describes the SPCOM fields.

Table 30-4. SPCOM Field Descriptions

Bits	Name	Description
0	STR	Start transmit. For an SPI master, setting STR causes the SPI to start transferring data to and from the Tx/Rx buffers if they are prepared. For a slave, setting STR when the SPI is idle causes it to load the Tx data register from the SPI Tx buffer and start sending with the next SPICLK after $\overline{\text{SPISEL}}$ is asserted. STR is cleared automatically after one system clock cycle.
1–7	—	Reserved and should be cleared.

30.5 SPI Parameter RAM

The SPI parameter RAM area begins at the SPI base address. It is similar to the SCC general-purpose parameter RAM. Some values must be user-initialized before the SPI is enabled; the CPM initializes the others. Once initialized, parameter RAM values do not usually need to be accessed. They should be changed only when the SPI is inactive. Table 30-5 shows the memory map of the SPI parameter RAM.

Table 30-5. SPI Parameter RAM Memory Map

Offset [1]	Name	Width	Description
0x00	**RBASE**	Hword	Rx/Tx BD table base address. Indicate where the BD tables begin in the dual-port RAM. Setting Rx/TxBD[W] in the last BD in each BD table determines how many BDs are allocated for the Tx and Rx sections of the SPI. Initialize RBASE/TBASE before enabling the SPI. Furthermore, do not configure BD tables of the SPI to overlap any other active controller's parameter RAM. RBASE and TBASE should be divisible by eight.
0x02	**TBASE**	Hword	

Table 30-5. SPI Parameter RAM Memory Map (continued)

Offset [1]	Name	Width	Description
0x04	**RFCR**	Byte	Rx/Tx function code. Contains the value to appear on AT[1-3] when the associated SDMA channel accesses memory. Also controls byte ordering for the transfers. See Section 30.5.1, “Receive/Transmit Function Code Registers (RFCR/TFCR).”
0x05	**TFCR**	Byte	
0x06	**MRBLR**	Hword	Maximum receive buffer length. The SPI has one MRBLR entry to define the maximum number of bytes the MPC855T writes to a Rx buffer before moving to the next buffer. The MPC855T can write fewer bytes than MRBLR if an error or end-of-frame occurs, but never exceeds the MRBLR value. User-supplied buffers should be no smaller than MRBLR. Tx buffers are unaffected by MRBLR and can have varying lengths; the number of bytes to be sent is programmed in TxBD[Data Length]. MRBLR is not intended to be changed while the SPI is operating. However it can be changed in a single bus cycle with one 16-bit move (not two 8-bit bus cycles back-to-back). The change takes effect when the CPM moves control to the next RxBD. To guarantee the exact RxBD on which the change occurs, change MRBLR only while the SPI receiver is disabled. MRBLR should be greater than zero; it should be an even number if the character length of the data exceeds 8 bits.
0x08	RSTATE	Word	Rx internal state. Reserved for CPM use.
0x0C	—	Word	The Rx internal data pointer [2] is updated by the SDMA channels to show the next address in the buffer to be accessed.
0x10	RBPTR	Hword	RxBD pointer. Points to the current Rx BD being processed or to the next BD to be serviced when idle. After a reset or when the end of the BD table is reached, the CPM initializes RBPTR to the RBASE value. Most applications should not modify RBPTR, but it can be updated when the receiver is disabled or when no Rx buffer is in use.
0x12	—	Hword	The Rx internal byte count[2] is a down-count value that is initialized with the MRBLR value and decremented with every byte the SDMA channels write.
0x14	—	Word	Rx temp. Reserved for CPM use.
0x18	TSTATE	Word	Tx internal state. Reserved for CPM use.
0x1C	—	Word	The Tx internal data pointer[2] is updated by the SDMA channels to show the next address in the buffer to be accessed.
0x20	TBPTR	Hword	TxBD pointer. Points to the current Tx BD during frame transmission or the next BD to be processed when idle. After reset or when the end of the Tx BD table is reached, the CPM initializes TBPTR to the TBASE value. Most applications do not need to modify TBPTR, but it can be updated when the transmitter is disabled or when no Tx buffer is in use.
0x22	—	Hword	The Tx internal byte count[2] is a down-count value initialized with TxBD[Data Length]and decremented with every byte read by the SDMA channels.
0x24	—	Word	Tx temp. Reserved for CPM use.
0x28 - 0x2F	—	—	Used during I^2C/SPI relocation, see Section 18.6.3, “Parameter RAM.

Note: The user must initialize only items in bold.

1 As programmed in SPI_BASE. The default value is IMMR + 0x3D80. See Section 18.6.3, “Parameter RAM.”
2 Normally, these parameters need not be accessed. They are listed to help experienced users in debugging.

30.5.1 Receive/Transmit Function Code Registers (RFCR/TFCR)

Figure 30-9 shows the fields in the receive/transmit function code registers (RFCR/TFCR)

Bit	0	1	2	3	4	5	6	7
Field	—			**BO**		**AT[1–3]**		
Reset	0000_0000							
R/W	R/W							
Addr	SPI Base + 04 (RFCR)/SPI Base + 05 (TFCR)							

Figure 30-9. Receive/Transmit Function Code Registers (RFCR/TFCR)

Table 30-6 describes the RFCR/TFCR fields.

Table 30-6. RFCR/TFCR Field Descriptions

Bits	Name	Description
0–	—	Reserved, should be cleared.
3–4	**BO**	Byte ordering. Set BO to select the required byte ordering for the buffer. If BO is changed on-the-fly, it takes effect at the beginning of the next frame or BD. See Appendix A, “Byte Ordering.” 00 Reserved 01 Modified little-endian. 1x Big-endian or true little-endian.
5–7	**AT[1–3]**	Address type 1–3. Contains the user-defined function code value used during the SDMA channel memory access. AT0 is always driven high to identify this channel access as a DMA-type access.

30.6 SPI Commands

Table 30-7 lists transmit/receive commands sent to the CPM command register (CPCR).

Table 30-7. SPI Commands

Command	Description
INIT TX PARAMETERS	Initializes all transmit parameters in the parameter RAM to their reset state and should be issued only when the transmitter is disabled. The INIT TX AND RX PARAMETERS command can also be used to reset both the Tx and Rx parameters.
CLOSE RXBD	Forces the SPI controller to close the current RxBD and use the next BD for subsequently received data. If the controller is not receiving data, no action is taken. Use this command to extract data from a partially full buffer.
INIT RX PARAMETERS	Initializes all receive parameters in the parameter RAM to their reset state. Should be issued only when the receiver is disabled. The INIT TX AND RX PARAMETERS command can also be used to reset both the Tx and Rx parameters.

30.7 The SPI Buffer Descriptor (BD) Table

As shown in Figure 30-10, buffer descriptors (BDs) are organized into separate Rx and Tx BD tables in dual-port RAM. The tables have the same basic configuration as the SCC and SMCs and form circular queues that determine the order buffers are transferred. The CPM uses BDs to confirm reception and transmission or to indicate error conditions so that the core knows buffers have been serviced. The buffers themselves can be placed in external memory or in any unused parameter area of the dual-port RAM.

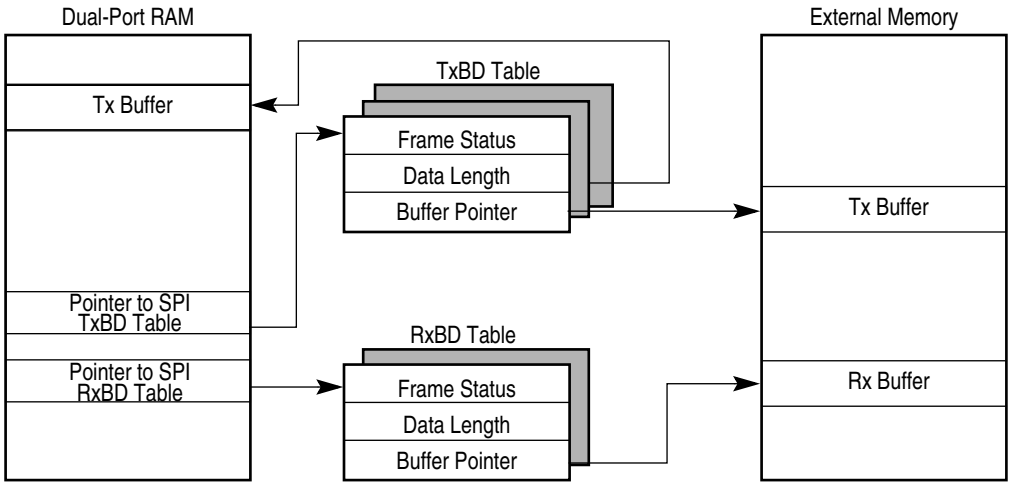


Figure 30-10. SPI Memory Structure

30.7.1 SPI Buffer Descriptors (BDs)

Receive and transmit BDs report information about each buffer transferred and whether a maskable interrupt should be generated. Each 64-bit BD, shown in Figure 30-11 and Figure 30-12, has the following structure:

- The half word at offset + 0 contains status and control bits. The CPM updates the status bits after the buffer is sent or received.
- The half word at offset + 2 contains the data length (in bytes) that is sent or received.
  - For an RxBD, this is the number of octets the CPM writes into this RxBD's buffer once the BD closes. The CPM updates this field after the received data is placed into the buffer. Memory allocated for this buffer should be no smaller than MRBLR.
  - For a TxBD, this is the number of octets the CPM should transmit from its buffer. Normally, this value should be greater than zero. If the character length is more than 8 bits, the data length should be even. For example, to send three characters of 8-bit data the data length field should be initialized to 3. However, to send 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 CPM never modifies this field.

- The word at offset + 4 points to the beginning of the buffer.
  - For an RxBD, the pointer must be even and can point to internal or external memory.
  - For a TxBD, the pointer can be even or odd, unless the character exceeds 8 bits, for which it must be even. The buffer can be in internal or external memory.

30.7.1.1 SPI Receive BD (RxBD)

The CPM uses RxBDs to report on each received buffer. It closes the current buffer, generates a maskable interrupt, and starts receiving data in the next buffer once the current buffer is full. The CPM also closes the buffer when the SPI is configured as a slave and $\overline{\text{SPISEL}}$ is negated, indicating that reception stopped. The core should write RxBD bits before the SPI is enabled. The format of an RxBD is shown in Figure 30-11.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**E**	—	**W**	**I**	L	—	**CM**	—							OV	ME
Offset + 2	Data Length															
Offset + 4	**Rx Buffer Pointer**															
Offset + 6																

Figure 30-11. SPI Receive BD (RxBD)

Table 30-8 describes the RxBD status and control fields.

Table 30-8. SPI RxBD Status and Control Field Descriptions

Bits	Name	Description
0	**E**	Empty. 0 The buffer is full or stopped receiving because of an error. The core can examine or write to any fields of this RxBD, but the CPM does not use this BD while E = 0. 1 The buffer is empty or reception is in progress. The CPM owns this RxBD and its buffer. Once E is set, the core should not write any fields of this RxBD.
1	—	Reserved, should be cleared.
2	**W**	Wrap (last BD in table). 0 Not the last BD in the RxBD table. 1 Last BD in the RxBD table. After this buffer is used, the CPM receives incoming data using the BD pointed to by RBASE (top of the table). The number of BDs in this table is determined only by the W bit.
3	**I**	Interrupt. 0 No interrupt is generated after this buffer is filled. 1 SPIE[RXB] is set when this buffer is full, indicating the need for the core to process the buffer. SPIE[RXB] causes an interrupt if not masked.
4	L	Last. Updated by the SPI when the buffer is closed. In slave mode, this occurs because $\overline{\text{SPISEL}}$ was negated. The SPI updates L after received data is placed in the buffer. 0 This buffer does not contain the last character of the message. 1 This buffer contains the last character of the message.
5	—	Reserved, should be cleared.

Table 30-8. SPI RxBD Status and Control Field Descriptions (continued)

Bits	Name	Description
6	**CM**	Continuous mode. Master mode only; in slave mode, CM should be cleared. 0 Normal operation. 1 The CPM does not clear RxBD[E] after this BD is closed; the buffer is overwritten when the CPM next accesses this BD. This allows continuous reception from an SPI slave into one buffer, which can be used, for example, for autoscanning of a serial A/D peripheral.
7–13	—	Reserved, should be cleared.
14	OV	Overrun. Set when a receiver overrun occurs during reception (slave mode only). The SPI updates OV after the received data is placed in the buffer.
15	ME	Multimaster error. Set when this buffer is closed because $\overline{\text{SPISEL}}$ was asserted when the SPI was in master mode. Indicates an arbitration problem between multiple masters on the SPI bus. The SPI updates ME after the received data is placed in the buffer.

30.7.1.2 SPI Transmit BD (TxBD)

Data to be sent with the SPI is sent to the CPM by arranging it in buffers referenced by TxBDs in the TxBD table. TxBD fields should be prepared before data is sent. The format of an TxBD is shown in Figure 30-12.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**R**	—	**W**	**I**	**L**	—	**CM**	—							UN	ME
Offset + 2	**Data Length**															
Offset + 4	**Tx Buffer Pointer**															
Offset + 6																

Figure 30-12. SPI Transmit BD (TxBD)

Table 30-9 describes the TxBD status and control fields.

Table 30-9. SPI TxBD Status and Control Field Descriptions

Bits	Name	Description
0	**R**	Ready. 0 The buffer is not ready to be sent. This BD or its buffer can be modified. The CPM clears R (unless RxBD[CM] is set) after the buffer is sent (unless RxBD[CM] is set) or an error occurs. 1 The buffer is ready for transmission or is being sent. The BD cannot be modified once R is set.
1	—	Reserved, should be cleared.
2	**W**	Wrap (last BD in TxBD table). 0 Not the last BD in the table. 1 Last BD in the table. After this buffer is used, the CPM receives incoming data using the BD pointed to by TBASE (top of the table). The number of BDs in this table is determined only by the W bit.
3	**I**	Interrupt. 0 No interrupt is generated after this buffer is processed if an error does not occur. 1 SPIE[TXB] or SPIE[TXE] are set when this buffer is processed and causes interrupts if not masked. Note that this bit does not mask SPIE[TXE].

Table 30-9. SPI TxBD Status and Control Field Descriptions (continued)

Bits	Name	Description
4	**L**	Last. 0 This buffer does not contain the last character of the message. 1 This buffer contains the last character of the message.
5	—	Reserved, should be cleared.
6	**CM**	Continuous mode. Valid only when the SPI is in master mode. In slave mode, it should be cleared. 0 Normal operation. 1 The CPM does not clear TxBD[R] after this BD is closed, allowing the buffer to be resent automatically when the CPM next accesses this BD.
7–13	—	Reserved, should be cleared.
14	UN	Underrun. Indicates that the SPI encountered a transmitter underrun condition while sending the buffer. This error occurs only when the SPI is in slave mode. The SPI updates UN after it sends the buffer.
15	ME	Multimaster error. Indicates that this buffer is closed because $\overline{\text{SPISEL}}$ was asserted when the SPI was in master mode. An arbitration problem occurred between devices on the SPI bus. The SPI updates ME after sending the buffer.

30.8 SPI Master Programming Example

The following sequence initializes the SPI to run at a high speed in master mode:

1. Configure port B to enable SPIMISO, SPIMOSI, and SPICLK. Set PBPAR[28, 29,30] and PBDIR[28, 29, 30], then clear PBODR[28, 29, 30].

   For multimaster operation, connect the internal $\overline{\text{SPISEL}}$ input to the SPI by setting PBPAR[31] and PBDIR[31] and by clearing PBODR[31].
2. Configure a parallel I/O signal to operate as the SPI select output signal if needed. If PB156 is chosen, clear PBODR[156] and PBPAR[156] and set PBDIR[156].
3. Write RBASE and TBASE in the SPI parameter RAM to point to the RxBD and TxBD tables in the dual-port RAM. Assuming one RxBD followed by one TxBD at the beginning of the dual-port RAM, write RBASE with 0x0000 and TBASE with 0x0008.
4. Execute the INIT RX AND TX PARAMETERS command by writing 0x0051 to CPCR.
5. Write 0x0001 to the SDCR to initialize the SDMA configuration register (SDCR).
6. Write RFCR and TFCR with 0x10 for normal operation.
7. Write MRBLR with the maximum number of bytes per Rx buffer. For this case, assume 16 bytes, so MRBLR = 0x0010.
8. Initialize the RxBD. Assume the Rx buffer is at 0x0000_1000 in main memory. Write 0xB000 to RxBD[Status and Control], 0x0000 to RxBD[Data Length] (optional), and 0x0000_1000 to RxBD[Buffer Pointer].
9. Initialize the TxBD. Assume the Tx buffer is at 0x0000_2000 in main memory and contains five 8-bit characters. Write 0xB800 to TxBD[Status and Control], 0x0005 to TxBD[Data Length], and 0x0000_2000 to TxBD[Buffer Pointer].

10. Write 0xFF to SPIE to clear any previous events.
11. Write 0x37 to SPIM to enable all possible SPI interrupts.
12. Write 0x0000_0020 to the CPM interrupt mask register (CIMR). This sets CIMR[SPI] to enable SPI-generated system interrupts. The CICR should also be initialized.
13. Write 0x0370 to SPMODE to enable normal operation (not loopback), master mode, SPI enabled, 8-bit characters, and the fastest speed possible.
14. Clear PBDAT[156], assuming PB156 is chosen above, to constantly assert the SPI select output signal.
15. Set SPCOM[STR] to start the transfer.

After 5 bytes are sent, the TxBD is closed because TxBD[L] is set. The RxBD is closed when the TxBD closes.

30.9 SPI Slave Programming Example

The following is an example initialization sequence to follow when the SPI is in slave mode. It is very similar to the SPI master example, except that $\overline{\text{SPISEL}}$ is used instead of a general-purpose I/O signal (as shown in Figure 30-2).

1. Set PBPAR[28–31] and PBDIR[28–31] to enable SPIMISO, SPIMOSI, SPICLK, and $\overline{\text{SPISEL}}$, then clear PBODR[28–31].
2. Assuming one RxBD at the beginning of the dual-port RAM followed by one TxBD, write RBASE with 0x0000 and TBASE with 0x0008 in the SPI parameter RAM.
3. Write RFCR and TFCR with 0x10 for normal operation.
4. Execute the INIT RX AND TX PARAMETERS command by writing 0x0051 to CPCR.
5. Write 0x0001 to SDCR.
6. Set MRBLR = 0x0010 for 16 bytes, the maximum number of bytes per buffer.
7. Initialize the RxBD. Assume the Rx buffer is at 0x0000_1000 in main memory. Write 0xB000 to RxBD[Status and Control], 0x0000 to RxBD[Data Length] (optional), and 0x0000_1000 to RxBD[Buffer Pointer].
8. Initialize the TxBD. Assume the Tx buffer is at 0x0000_2000 in main memory and contains five 8-bit characters. Write 0xB800 to TxBD[Status and Control], 0x0005 to TxBD[Data Length], and 0x0000_2000 to TxBD[Buffer Pointer].
9. Write 0xFF to SPIE to clear any previous events.
10. Write 0x37 to SPIM to enable all SPI interrupts.
11. Write 0x0000_0020 to CIMR (that is, set CIMR[SPI]) to allow the SPI to generate a system interrupt. The CICR should also be initialized.

12. Set SPMODE to 0x0170 to enable normal operation (not loopback), slave mode, SPI enabled, and 8-bit characters. Baud-rate generator speed is ignored in slave mode.
13. Set SPCOM[STR] to enable the SPI to be ready once the master begins the transfer.

Note that if the master sends 3 bytes and negates $\overline{\text{SPISEL}}$, the RxBD is closed but the TxBD remains open. If the master sends 5 or more bytes, the TxBD is closed after the fifth byte. If the master sends 16 bytes and negates $\overline{\text{SPISEL}}$, the RxBD is closed without triggering a busy error (SPIE[BSY]). If the master sends more than 16 bytes, the RxBD is closed (full) and an SPIE[BSY] event occurs after the 17th byte is received.

30.10 Handling Interrupts in the SPI

The following sequence should be followed to handle interrupts in the SPI:

1. Once an interrupt occurs, read SPIE to determine the interrupt source. Normally, SPIE bits should be cleared at this time.
2. Process the TxBD to reuse it and the RxBD to extract the data from it. To transmit another buffer, simply set TxBD[R], RxBD[E], and SPCOM[STR].
3. Clear the interrupt by writing a one to CISR[SPI].
4. Execute an **rfi** instruction.

Chapter 31
I²C Controller

The inter-integrated circuit (I²C®) controller lets the MPC855T exchange data with other I²C devices, such as microcontrollers, EEPROMs, real-time clock devices, A/D converters, and LCD displays. The I²C controller uses a synchronous, multimaster bus that can connect several integrated circuits on a board. It uses two signals—serial data (SDA) and serial clock (SCL)—to carry information between the integrated circuits connected to it.

As shown in Figure 31-1, the I²C controller consists of transmit and receive sections, an independent baud-rate generator (BRG), and a control unit. The transmit and receive sections use the same clock, which is derived from the I²C BRG when in master mode and generated externally when in slave mode. Wait states are inserted during a data transfer if SCL is held low by a slave device. In the middle of a data transfer, the master I²C controller recognizes the need for wait states by monitoring SCL. However, the I²C controller has no automatic time-out mechanism if the slave device does not release SCL; therefore, software should monitor how long SCL stays low to generate bus timeouts.

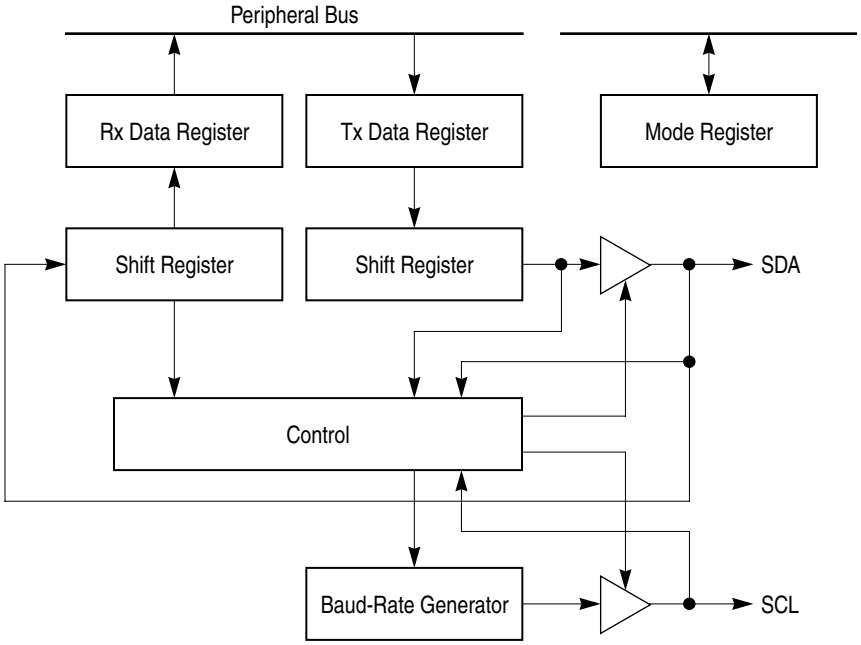


Figure 31-1. I²C Controller Block Diagram

The I²C receiver and transmitter are double-buffered, which corresponds to an effective two-character FIFO latency. In normal operation, the msb (bit 0) is shifted out first. When the I²C is not enabled in the I²C mode register (I2MOD[EN] = 0), it consumes little power.

31.1 I²C Features

The following is a list of the I²C controller's main features:

- Two-signal interface (SDA on PB[27] and SCL on PB[26])
- Support for master and slave I²C operation
- Multiple-master environment support
- Continuous transfer mode for autoscanning of a peripheral
- Supports a maximum clock rate of 520 KHz (with a CPM utilization of 25%), assuming a 25-MHz system clock.
- Independent, programmable baud-rate generator
- Supports 7-bit I²C addressing
- Open-drain output signals allow multiple master configuration
- Local loopback capability for testing

31.2 I²C Controller Clocking and Signal Functions

The I²C controller can be configured as a master or slave for the serial channel. As a master, the controller's BRG provides the transfer clock. The I²C BRG takes its input from the BRG clock (BRGCLK), which is described in Section 14.3, "Clock Signals."

SDA and SCL are bidirectional signals connected to a positive supply voltage through an external pull-up resistor. When the bus is free, both signals are pulled high. The general I²C master/slave configuration is shown in Figure 31-2.

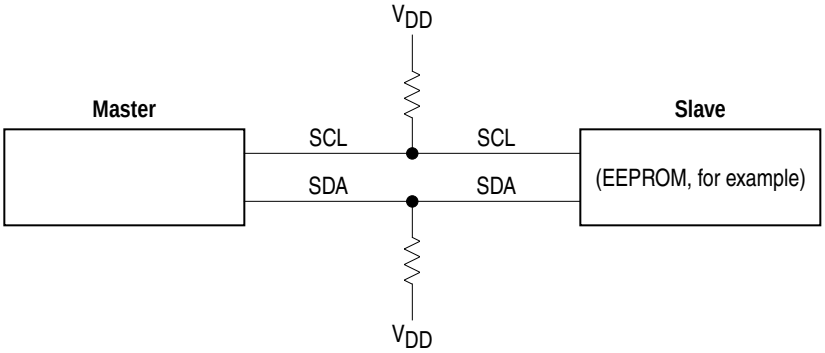


Figure 31-2. I²C Master/Slave General Configuration

When the I²C controller is the master, the SCL clock output, taken directly from the I²C BRG, shifts receive data in and transmit data out through SDA. The transmitter arbitrates for the bus during transmission and aborts if it loses arbitration. When the I²C controller is

a slave, the SCL clock input shifts data in and out through SDA. The SCL frequency can range from DC to BRGCLK/48.

31.3 I²C Controller Transfers

To initiate a transfer, the master I²C controller sends a message specifying a read or write request to an I²C slave. The first byte of the message consists of a 7-bit slave port address and a R/W request bit. Note that because the R/W request follows the slave port address in the I²C bus specification, the R/W request bit must be placed in the lsb (bit 7) unless operating in reverse data mode; see Section 31.4.1, “I2C Mode Register (I2MOD).”

To write to a slave, the master sends a write request (R/W = 0) along with either the target slave’s address or the general call (broadcast) address of all zeros, followed by the data to be written. To read from a slave, the master sends a read request (R/W = 1) and the target slave’s address. When the target slave acknowledges the read request, the transfer direction is reversed, and the master receives the slave’s transmit buffers. If the receiver (master or slave) does not acknowledge each byte transfer in the ninth bit frame, the transmitter signals a transmission error event (I2ER[TXE]). An I²C transfer timing diagram is shown in Figure 31-3.

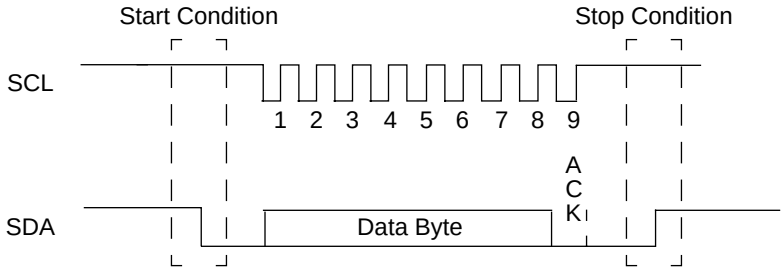


Figure 31-3. I²C Transfer Timing

Select master or slave mode for the controller using the I²C command register (I2COM[M/S]). Set the master’s start bit, I2COM[STR], to begin a transfer; setting a slave’s I2COM[STR] activates the slave to wait for a transfer request from a master.

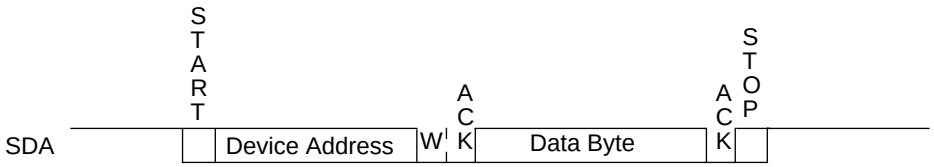
If a master or slave transmitter’s current TxBD[L] is set, transmission stops once the buffer is sent; that is, I2COM[STR] must be set again to reactivate transfers. If TxBD[L] is zero, once the current buffer is sent, the controller begins processing the next TxBD without waiting for I2COM[STR] to be set again.

The following sections further detail the transfer process.

31.3.1 I²C Master Write (Slave Read)

If the MPC855T is the master, prepare the transmit buffers and BDs before initiating a write. Initialize the first transmit data byte with the slave address and write request (R/W = 0).

If the MPC855T is the slave target of the write, prepare receive buffers and BDs to await the master's request. Figure 31-4 shows the timing for a master write.



Note: Data and ACK are repeated *n* times.

Figure 31-4. I²C Master Write Timing

A master write performed by the MPC855T occurs as follows:

1. Set the master's I2COM[STR]. The transfer starts when the SDMA channel loads the transmit FIFO with data and the I²C bus is not busy.
2. The I²C master generates a start condition—a high-to-low transition on SDA while SCL is high—and the transfer clock SCL pulses for each bit shifted out on SDA. If the master transmitter detects a multiple-master collision (by sensing a '0' on SDA while sending a '1'), transmission stops and the channel reverts to slave mode. A maskable interrupt is sent to the master's core so software can try to retransmit later.
3. The slave acknowledges each byte and writes to its current receive buffer until a new start or stop condition is detected.
4. After sending each byte, the master monitors the acknowledge indication. If the slave receiver fails to acknowledge a byte, transmission stops and the master generates a stop condition—a low-to-high transition on SDA while SCL is high.

31.3.2 I²C Loopback Testing

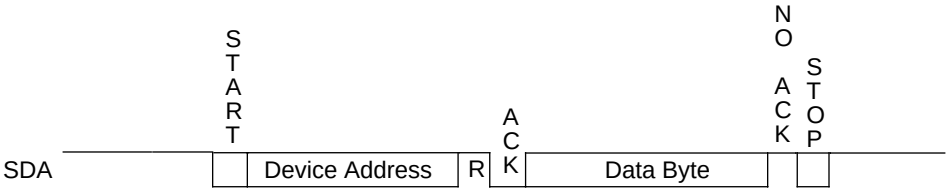
When in master mode, an I²C controller supports loopback operation for master write requests. The master I²C controller simply issues a write request directed to its own address (programmed in I2ADD). The master's receiver monitors the transmission and reads the transmitted data into its receive buffer. Loopback operation requires no special register programming.

31.3.3 I²C Master Read (Slave Write)

Before initiating a master read with the MPC855T, prepare a transmit buffer of size **n**+1 bytes, where **n** is the number of bytes to be read from the slave. The first transmit byte

should be initialized to the slave address with R/W = 1. The next **n** transmit bytes are used strictly for timing and can be left uninitialized. Configure suitable receive buffers and BDs to receive the slave's transmission.

If the MPC855T is the slave target of the read, prepare the I²C transmit buffers and BDs and activate it by setting I2COM[STR]. Figure 31-5 shows the timing for a master read.



Note: After the *n*th data byte, the master does not acknowledge the slave.

Figure 31-5. I²C Master Read Timing

A master read performed by the MPC855T occurs as follows:

1. Set the master's I2COM[STR] to initiate the read. The transfer starts when the SDMA channel loads the transmit FIFO with data and the I²C bus is not busy.
2. The slave detects a start condition on SDA and SCL.
3. After the first byte is shifted in, the slave compares the received data to its slave address. If the slave is an MPC855T, the address is programmed in its I²C address register (I2ADD).
   - If a match is found and the slave is ready, then the slave acknowledges the request and begins sending on the clock pulse after the acknowledge. If the slave is an MPC855T, it is ready when its transmit FIFO has been loaded by the SDMA channel (the transmit buffers and BDs have been prepared and I2COM[STR] has been set).
   - If a match is found but the slave is not ready, the read request is not acknowledged and the transaction is aborted. If the slave is an MPC855T, a maskable transmission error interrupt is triggered to allow software to prepare data for transmission on the next try.
   - If a mismatch occurs, the slave ignores the message and searches for a new start condition.
4. The master acknowledges each byte sent as long as an overrun does not occur. If the master receiver fails to acknowledge a byte, the slave aborts transmission. For a slave MPC855T, the abort generates a maskable interrupt. A maskable interrupt is also issued after a complete buffer is sent or after an error. If an underrun occurs, the MPC855T slave sends ones until a stop condition is detected.

31.3.4 I²C Multi-Master Considerations

The I²C controller supports a multi-master configuration, in which the I²C controller must alternate between master and slave modes. The I²C controller supports this by implementing I²C master arbitration in hardware. However, due to the nature of the I²C bus and the implementation of the I²C controller, certain software considerations must be made.

An MPC855T I²C controller attempting a master read request could simultaneously be targeted for an external master write (slave read). Both operations trigger the controller's I2CER[RXB] event, but only one operation wins the bus arbitration. To determine which operation caused the interrupt, software must verify that its transmit operation actually completed before assuming that the received data is the result of its read operation.

Problems could also arise if the MPC855T's I²C controller master sets up a transmit buffer and BD for a write request, but then is the target of a read request from another master. Without software precautions, the I²C controller responds to the other master with the transmit buffer originally intended for its own write request. To avoid this situation, a higher-level handshake protocol must be used. For example, a master, before reading a slave, writes the slave with a description of the requested data (which register should be read, for example). This operation is typical with many I²C devices.

31.4 I²C Registers

The following sections describe the I²C registers.

31.4.1 I²C Mode Register (I2MOD)

The I²C mode register, shown in Figure 31-6, controls the I²C modes and clock source.

Bit	0	1	2	3	4	5	6	7
Field	—		REVD	GCD	FLT	PDIV		EN
Reset	0000_0000							
R/W	R/W							
Addr	0x860							

Figure 31-6. I²C Mode Register (I2MOD)

This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$. Table 31-1 describes I2MOD bit functions.

Table 31-1. I2MOD Field Descriptions

Bits	Name	Description
0–1	—	Reserved and should be cleared.
2	REVD	Reverse data. Determines the Rx and Tx character bit order. 0 Normal operation. The msb (bit 0) of each character is sent and received first. 1 Reverse data. The lsb (bit 7) of each character is sent and received first. Note: Clearing REVD is strongly recommended to ensure consistent bit ordering across devices.
3	GCD	General call disable. Determines whether the receiver acknowledges a general call address (all zeros). 0 General call address is enabled. 1 General call address is disabled.
4	FLT	Clock filter. Determines if the I^{2}C input clock SCL is filtered to prevent spikes in a noisy environment. 0 SCL is not filtered. 1 SCL is filtered by a digital filter.
5–6	PDIV	Predivider. Selects the clock division factor before it is input into the I^{2}C BRG. The clock source for the I^{2}C BRG is the BRGCLK generated by the SIU. 00 BRGCLK/32 01 BRGCLK/16 10 BRGCLK/8 11 BRGCLK/4 Note: To both save power and reduce noise susceptibility, select the PDIV with the largest division factor (slowest clock) that still meets performance requirements.
7	EN	Enable I^{2}C operation. 0 I^{2}C is disabled. The I^{2}C is in a reset state and consumes minimal power. 1 I^{2}C is enabled. Do not change other I2MOD bits when EN is set.

31.4.2 I^{2}C Address Register (I2ADD)

The I^{2}C address register, shown in Figure 31-7, holds the address for this I^{2}C port.

Bit	0	1	2	3	4	5	6	7
Field	SAD							—
Reset	Undefined							
R/W	R/W							
Addr	0x864							

Figure 31-7. I^{2}C Address Register (I2ADD)

This register is not affected by $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$. Table 31-2 describes I2CADD fields.

Table 31-2. I2ADD Field Descriptions

Bits	Name	Description
0–6	SAD	Slave address 0–6. Holds the slave address for the I^{2}C port.
7	—	Reserved and should be cleared.

31.4.3 I²C Baud Rate Generator Register (I2BRG)

The I²C baud rate generator register, shown in Figure 31-8, sets the divide ratio of the I²C BRG.

Bit	0	1	2	3	4	5	6	7
Field	DIV							
Reset	1111_1111							
R/W	R/W							
Addr	0x868							

Figure 31-8. I²C Baud Rate Generator Register (I2BRG)

This register is affected by $\overline{\text{HRESET}}$ but is not affected by $\overline{\text{SRESET}}$. Table 31-3 describes I2BRG fields.

Table 31-3. I2BRG Field Descriptions

Bits	Name	Description
0–7	DIV	Division ratio 0–7. Specifies the divide ratio of the BRG divider in the I²C clock generator. The output of the prescaler is divided by 2 * ([DIV0–DIV7] + 3) and the clock has a 50% duty cycle. DIV must be programmed to a minimum value of 3 if the digital filter is disabled and 6 if it is enabled.

31.4.4 I²C Event/Mask Registers (I2CER/I2CMR)

The I²C event register (I2CER) is used to generate interrupts and report events. When an event is recognized, the I²C controller sets the corresponding I2CER bit. I2CER bits are cleared by writing ones—writing zeros has no effect. Setting a bit in the I²C mask register (I2CMR) enables and clearing a bit masks the corresponding interrupt. Unmasked I2CER bits must be cleared before the CPM clears internal interrupt requests. Figure 31-9 shows both registers.

Bit	0	1	2	3	4	5	6	7
Field	—			TXE	—	BSY	TXB	RXB
Reset	0000_0000							
R/W	R/W							
Addr	0x870(I2CER)/0x874 (I2CMR)							

Figure 31-9. I²C Event/Mask Registers (I2CER/I2CMR)

This registers are affected by HRESET and SRESET. Table 31-4 describes the I2CER/I2CMR fields.

Table 31-4. I2CER/I2CMR Field Descriptions

Bits	Name	Description
0–2	—	Reserved and should be cleared.
3	TXE	Tx error. Set when an error occurs during transmission.
4	—	Reserved and should be cleared.
5	BSY	Busy. Set after the first character is received but discarded because no Rx buffer is available.
6	TXB	Tx buffer. Set when the Tx data of the last character in the buffer has been sent.
7	RXB	Rx buffer. Set after the last character is written to the Rx buffer and the RxBD is closed.

31.4.5 I²C Command Register (I2COM)

The I²C command register, shown in Figure 31-10, is used to start I²C transfers and to select master or slave mode.

Bit	0	1	2	3	4	5	6	7
Field	STR	—						M/S
Reset	0000_0000							
R/W	R/W							
Addr	0x86C							

Figure 31-10. I²C Command Register (I2COM)

This register is affected by HRESET and SRESET. Table 31-5 describes I2COM fields.

Table 31-5. I2COM Field Descriptions

Bits	Name	Description
0	STR	Start transmit. In master mode, setting STR causes the I²C controller to start sending data from the I²C Tx buffers if they are ready. In slave mode, setting STR when the I²C controller is idle causes it to load the Tx data register from the current Tx buffer (if ready) and start sending when it receives an address byte that matches the slave address with R/W = 1. STR is always read as a 0.
1–6	—	Reserved and should be cleared.
7	M/S	Master/slave. Configures the I²C controller to operate as a master or a slave. 0 I²C is a slave. 1 I²C is a master.

31.5 I²C Parameter RAM

The I²C controller parameter RAM area, shown in Table 31-6, is used for the general I²C parameters. It is similar to the SCC general-purpose parameter RAM. Certain parameter RAM values, such as the maximum receive buffer length (MRBLR), must be initialized by

the user before the I²C is enabled, while other parameters, used by the CPM, do not need initialization. Software usually does not access parameter RAM entries once they are initialized; they should be changed only when the I²C is inactive.

Table 31-6. I²C Parameter RAM Memory Map

Offset [1]	Name	Width	Description
0x00	**RBASE**	Hword	Rx/TxBD table base address. Indicate where the BD tables begin in the dual-port RAM. Setting Rx/TxBD[W] in the last BD in each BD table determines how many BDs are allocated for the Tx and Rx sections of the I²C. Initialize RBASE/TBASE before enabling the I²C. Furthermore, do not configure BD tables of the I²C to overlap any other active controller's parameter RAM. RBASE and TBASE should be divisible by eight.
0x02	**TBASE**	Hword	
0x04	**RFCR**	Byte	Rx/Tx function code. The value to appear on AT[1–3] when the associated SDMA channel accesses memory. Also controls the byte-ordering convention for transfers. See Figure 31-11 and Table 31-7.
0x05	**TFCR**	Byte	
0x06	**MRBLR**	Hword	Maximum receive buffer length. Defines the maximum number of bytes the I²C receiver writes to a receive buffer before moving to the next buffer. The receiver writes fewer bytes to the buffer than the MRBLR value if an error or end-of-frame occurs. Receive buffers should not be smaller than MRBLR. Transmit buffers are unaffected by MRBLR and can vary in length; the number of bytes to be sent is specified in TxBD[Data Length]. MRBLR is not intended to be changed while the I²C is operating. However it can be changed in a single bus cycle with one 16-bit move (not two 8-bit bus cycles back-to-back). The change takes effect when the CP moves control to the next RxBD. To guarantee the exact RxBD on which the change occurs, change MRBLR only while the I²C receiver is disabled. MRBLR should be greater than zero.
0x08	RSTATE	Word	Rx internal state. Reserved for CPM use.
0x0C	RPTR	Word	Rx internal data pointer [2] is updated by the SDMA channels to show the next address in the buffer to be accessed.
0x10	RBPTR	Hword	RxBD pointer. Points to the next descriptor the receiver transfers data to when it is in an idle state or to the current descriptor during frame processing for each I²C channel. After a reset or when the end of the descriptor table is reached, the CP initializes RBPTR to the value in RBASE. Most applications should not write RBPTR, but it can be modified when the receiver is disabled or when no receive buffer is used.
0x12	RCOUNT	Hword	Rx internal byte count [2] is a down-count value that is initialized with the MRBLR value and decremented with every byte the SDMA channels write.
0x14	RTEMP	Word	Rx temp. Reserved for CPM use.
0x18	TSTATE	Word	Tx internal state. Reserved for CPM use.
0x1C	TPTR	Word	Tx internal data pointer [2] is updated by the SDMA channels to show the next address in the buffer to be accessed.
0x20	TBPTR	Hword	TxBD pointer. Points to the next descriptor that the transmitter transfers data from when it is in an idle state or to the current descriptor during frame transmission. After a reset or when the end of the descriptor table is reached, the CPM initializes TBPTR to the value in TBASE.Most applications should not write TBPTR, but it can be modified when the transmitter is disabled or when no transmit buffer is used.
0x22	TCOUNT	Hword	Tx internal byte count [2] is a down-count value initialized with TxBD[Data Length] and decremented with every byte read by the SDMA channels.

Table 31-6. I²C Parameter RAM Memory Map (continued)

Offset [1]	Name	Width	Description
0x24	TTEMP	Word	Tx temp. Reserved for CP use.
0x28-0x2F	—	—	Used for I²C/SPI relocation, see Section 18.6.3, “Parameter RAM.

1 As programmed in I2C_BASE. The default value is IMMR + 0x3C80. See Section 18.6.3, “Parameter RAM.”
2 Normally, these parameters need not be accessed.

Figure 31-11 shows the RFCR/TFCR bit fields.

Bit	0	1	2	3	4	5	6	7
Field	—			BO		AT[1–3]		
Reset	0000_0000							
R/W	R/W							
Addr	I2C Base + 04 (RFCR)/I2C Base + 05 (TFCR)							

Figure 31-11. I²C Function Code Registers (RFCR/TFCR)

Table 31-7 describes the RFCR/TFCR bit fields.

Table 31-7. RFCR/TFCR Field Descriptions

Bits	Name	Description
0–2	—	Reserved, should be cleared.
3–4	**BO**	Byte ordering. Set BO to select the required byte ordering for the buffer. If BO is changed on-the-fly, it takes effect at the beginning of the next frame (Ethernet, HDLC, and transparent) or at the beginning of the next BD. See Appendix A, “Byte Ordering.” 00 Reserved 01 Modified little-endian. 1x Big-endian or true little-endian.
5–7	**AT[1–3]**	Address type 1–3. Contains the user-defined function code value used during the SDMA channel memory access. AT0 is always driven high to identify this channel access as a DMA-type access.

31.6 I²C Commands

The I²C transmit and receive commands, shown in Table 31-8, are issued to the CPM command register (CPCR).

Table 31-8. I²C Transmit/Receive Commands

Command	Description
INIT TX PARAMETERS	Initializes all transmit parameters in the parameter RAM to their reset state. Should be issued only when the transmitter is disabled. The INIT TX AND RX PARAMETERS command can also be used to reset both the Tx and Rx parameters.

Table 31-8. I²C Transmit/Receive Commands (continued)

Command	Description
CLOSE RXBD	Forces the I²C controller to close the current Rx BD and use the next BD for subsequently received data. If the controller is not receiving data, no action is taken. Use this command to extract data from a partially full buffer.
INIT RX PARAMETERS	Initializes all receive parameters in the parameter RAM to their reset state. Should be issued only when the receiver is disabled. The INIT TX AND RX PARAMETERS command can also be used to reset both the Tx and Rx parameters.

31.7 I²C Buffer Descriptor (BD) Tables

As shown in Figure 31-12, buffer descriptors (BDs) are organized into separate Rx and TxBD tables in dual-port RAM. The tables have the same basic configuration as for the SCC and SMCs and form circular queues that determine the order buffers are transferred. The CPM uses BDs to confirm reception and transmission or to indicate error conditions so that the core knows buffers have been serviced. The buffers themselves can be placed in external memory or in any unused parameter area of the dual-port RAM.

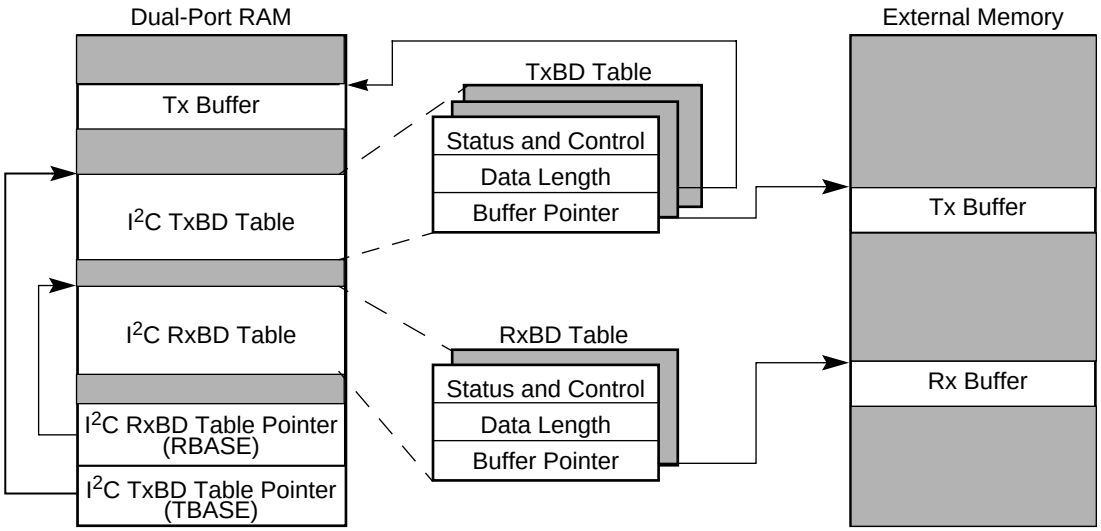


Figure 31-12. I²C Memory Structure

31.7.1 I²C Buffer Descriptors (BDs)

Receive and transmit buffer descriptors report information about each buffer transferred and whether a maskable interrupt should be generated. Each 64-bit BD, shown in Figure 31-13 and Figure 31-14, has the following structure:

- The half word at offset + 0 contains status and control bits. The CPM updates the status bits after the buffer is sent or received.
- The half word at offset + 2 contains the data length (in bytes) that is sent or received.

— For an RxBD, this is the number of octets the CPM writes into this RxBD's buffer once the descriptor closes. The CPM updates this field after the received data is placed into the associated buffer. Memory allocated for this buffer should be no smaller than MRBLR.

— For a TxBD, this is the number of octets the CPM should transmit from its buffer. Normally, this value should be greater than zero. The CPM never modifies this field.

- The word at offset + 4 points to the beginning of the buffer.

— For an RxBD, the pointer must be even and can point to internal or external memory.

— For a TxBD, the pointer can be even or odd. The buffer can reside in internal or external memory.

31.7.1.1 I²C Receive Buffer Descriptor (RxBD)

Using RxBDs, the CPM reports on each buffer received, closes the current buffer, generates a maskable interrupt, and starts receiving data in the next buffer when the current one is full. It closes the buffer when a stop or start condition is found on the I²C bus or when an overrun error occurs. The core should write RxBD bits before the I²C controller is enabled.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**E**	—	**W**	**I**	L	—									OV	—
Offset + 2	Data Length															
Offset + 4	**RX Buffer Pointer**															
Offset + 6																

Figure 31-13. I²C Receive Buffer Descriptor (RxBD)

Table 31-9 describes I²C RxBD status and control bits.

Table 31-9. I²C RxBD Status and Control Bits

Bits	Name	Description
0	**E**	Empty. 0 The buffer is full or stopped receiving because of an error. The core can examine or write to any fields of this RxBD, but the CPM does not use this BD while E = 0. 1 The buffer is empty or reception is in progress. The CPM owns this RxBD and its buffer. Once E is set, the core should not write any fields of this RxBD.
1	—	Reserved and should be cleared.
2	**W**	Wrap (last BD in table). 0 Not the last BD in the RxBD table. 1 Last BD in the RxBD table. After this buffer is used, the CPM receives incoming data using the BD pointed to by RBASE (top of the table). The number of BDs in this table is determined only by the W bit.

Table 31-9. I²C RxBD Status and Control Bits (continued)

Bits	Name	Description
3	**I**	Interrupt. 0 No interrupt is generated after this buffer is full. 1 The I2CER[RXB] is set when the CPM fills this buffer, indicating that the core needs to process the buffer. The RXB bit can cause an interrupt if it is enabled.
4	L	Last. The I²C controller sets L. 0 This buffer does not contain the last character of the message. 1 This buffer holds the last character of the message. The I²C controller sets L after all received data is placed into the associated buffer, or because of a stop or start condition or an overrun.
5–13	—	Reserved and should be cleared.
14	OV	Overrun. Set when a receiver overrun occurs during reception. The I²C controller updates this bit after the received data is placed into the associated buffer.
15	—	Reserved and should be cleared.

31.7.1.2 I²C Transmit Buffer Descriptor (TxBD)

Transmit data is arranged in buffers referenced by TxBDs in the TxBD table. The first word of the TxBD, shown in Figure 31-14, contains status and control bits.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**R**	—	**W**	**I**	**L**	**S**	—							NAK	UN	CL
Offset + 2	**Data Length**															
Offset + 4	**Tx Buffer Pointer**															
Offset + 6																

Figure 31-14. I²C Transmit Buffer Descriptor (TxBD)

Table 31-10 describes I²C TxBD status and control bits.

Table 31-10. I²C TxBD Status and Control Bits

Bits	Name	Description
0	**R**	Ready. 0 The buffer is not ready to be sent. This BD or its buffer can be modified. The CPM clears R after the buffer is sent or an error occurs. 1 The buffer is ready for transmission or is being sent. The BD cannot be modified once R is set.
1	—	Reserved and should be cleared.
2	**W**	Wrap (last BD in TxBD table). 0 Not the last BD in the table. 1 Last BD in the table. After this buffer is used, the CPM transmits data using the BD pointed to by TBASE (top of the table). The number of BDs in this table is determined only by the W bit.
3	**I**	Interrupt. 0 No interrupt is generated after this buffer is serviced. 1 I2CER[TXB] or I2CER[TXE] is set when the buffer is serviced. If enabled, an interrupt occurs.

Table 31-10. I²C TxBD Status and Control Bits (continued)

Bits	Name	Description
4	**L**	Last. 0 This buffer does not contain the last character of the message. 1 This buffer contains the last character of the message. After sending this buffer, the transmitter generates a stop condition and deactivates. (Retrigger I2COM[STR] to initiate a new transmission.)
5	**S**	Generate start condition. Provides ability to send back-to-back messages on one I2COM[STR] trigger. 0 Do not send a start condition before the first byte of the buffer. 1 Send a start condition before the first byte of the buffer. (Used to separate messages.) Note: If this BD is the first one in a message when I2COM[STR] is triggered, a start condition is sent regardless of the value of TxBD[S].
6–12	—	Reserved and should be cleared.
13	NAK	No acknowledge. Indicates that the transmission was aborted because the last byte sent was not acknowledged. The I²C controller updates NAK after the buffer is sent.
14	UN	Underrun. Indicates that the I²C controller encountered a transmitter underrun condition while sending the associated buffer. The I²C controller updates UN after the buffer is sent.
15	CL	Collision. Indicates that transmission terminated because the transmitter was lost while arbitrating for the bus. The I²C controller updates CL after the buffer is sent.

Chapter 32
Parallel Interface Port (PIP)

Multiplexed through the 18-bit port B parallel I/O, the parallel interface port (PIP) allows data to be sent to and from the MPC855T over 8 or 16 parallel data lines with two handshake control signals. The PIP signals are grouped into two sets, PB[24–31] and PB[14–23], allowing the PIP to be configured as an 8- or 16-bit port. When the PIP is used, SMC2 is not available since they share registers and parameter RAM.

PIP transfers can operate in one of the three following modes:

- Handshaking I/O port. Timing attributes, such as setup time and pulse width, are programmable. Controlled by the CP or the core. There are two handshake options for strobed I/O:
  - Two interlocked handshake signals. Supports level-sensitive handshake control signals compatible with the advanced byte transfer mode of the P1284 protocol; see Section 32.7.1, “Interlocked Handshake Mode.”
  - Two pulsed handshake signals. Supports edge-sensitive handshakes like those used for a Centronics interface; see Section 32.7.2, “Pulsed Handshake Mode.”
- Transparent I/O port with one strobe signal. Controlled by the CP only. See Section 32.8, “Transparent Transfers.”

32.1 Features

The following is a list of the PIP’s main features:

- Eighteen general-purpose I/O signals
- Two handshake modes for strobed I/O
- Transparent I/O using a single strobe
- Programmable handshake timing attributes
- Supports the Centronics receiver/transmitter interface
- Supports fast connection between MPC860 family devices
- Can be controlled by the core or CP

Figure 32-1 is a block diagram of the PIP.

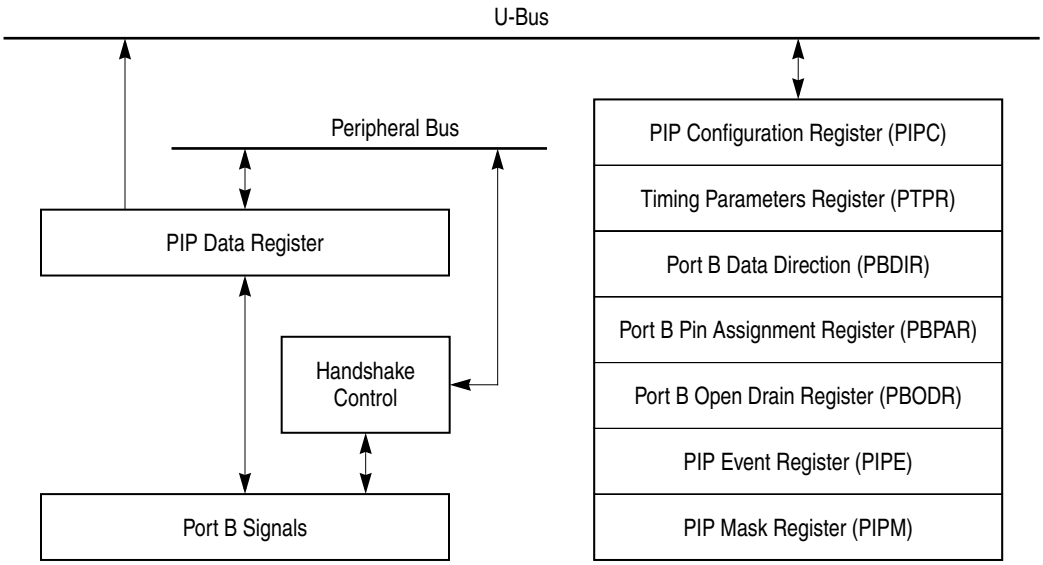


Figure 32-1. PIP Block Diagram

32.2 Core Control vs. CP Control

The host-control bit in the PIP configuration register PIPC[HSC] determines whether the PIP transfer is controlled by the CP or by the core.

32.2.1 Core Control

When the PIP is controlled by the core (PIPC[HSC] = 1), only the interlocked and pulsed handshaking modes can be used. The CP does not directly participate in the transfer, but an Rx or Tx character event (RCH or TCH) is flagged in the event register (PIPE). The PIPE is then masked against the PIP mask register (PIPM); unmasked events interrupt the core.

- When the PIP is configured to receive and $\overline{\text{STB}}$ is asserted on STBI (strobe-in on PB14), PIPE[RCH] is set to indicate that a character has arrived. When software reads the port B data register (PBDAT), the PIP asserts $\overline{\text{ACK}}$ through STBO (strobe-out on PB15).
- When the PIP is configured to send and the core writes PBDAT, $\overline{\text{STB}}$ is driven low on STBO (strobe-out on PB15). When the destination device drives $\overline{\text{ACK}}$ low onto STBI (strobe-in on PB14), the PIP indicates that a character was successfully sent by flagging PIPE[TCH].

For a core-controlled PIP, only the PIPC, PIPE, PIPM and port B registers need to be configured or monitored. The PIP parameter RAM and buffer descriptors are not used.

32.2.2 CP Control

When the PIP is controlled by the CP (PIPC[HSC] = 0), any of the three handshake modes

can be used. Data is prepared by the core using PIP buffer descriptors. CP-controlled strobed transfers are the same as core-controlled transfers described above, except reads and writes to PBDAT are done automatically by DMA. Blocks of data can be transferred without interrupting the core. The data block can span several linked buffers (a buffer chain), and an entire block can be transferred without core intervention. CP-controlled transparent transfers are described in Section 32.8, “Transparent Transfers.”

For a CP-controlled PIP:

- Initialize the PIPC and parameter RAM to configure the channel.
- Set up buffer descriptors and buffers for the DMA.
- Use PIPM and PIPE to control and monitor events reported.

32.3 The PIP Parameter RAM

The PIP remaps the SMC2 parameter RAM. The following subsections describe the PIP parameter RAM for sending and receiving.

32.3.1 PIP Transmitter Parameter RAM

The PIP transmitter uses the parameter RAM mapping shown in Table 32-1. Certain parameter RAM values must be initialized before the transmitter is enabled; others are initialized or written by the CP. Most software does not need access to parameter RAM values after initialization because activity centers around the buffer descriptors.

Table 32-1. PIP Transmitter Parameter RAM Memory Map

Offset[1]	Name	Width	Description
0x00	—	Hword	Reserved for receiving.
0x02	**TBASE**	Hword	PIP TxBD table base offset from the beginning of dual-port RAM. Initialize TBASE before enabling the channel. TBASE should be divisible by 8.
0x04	**PFCR**	Byte	PIP function code. Appears on AT[1-3] when the associated SDMA channel accesses memory. Also controls byte ordering for the transfers. See Section 32.3.1.1, “PIP Function Code Register (PFCR).”
0x05	**SMASK**	Byte	Status mask. Controls which, if any, printer status lines are checked before each transfer. See Section 32.3.1.2, “Status Mask Register (SMASK).”
0x06–0x17	—	—	Reserved for receiving.
0x18	TSTATE	Word	Tx internal state.
0x1C	T_PTR	Word	Tx internal data pointer.
0x20	TBPTR	Hword	TxBD pointer. Points to the current Tx BD during frame transmission or the next BD to be processed when idle. After reset or when the end of the Tx BD table is reached, the CP initializes TBPTR to the TBASE value. Most applications do not need to write TBPTR, but it can be updated when the transmitter is disabled or when no Tx buffer is in use.

Table 32-1. PIP Transmitter Parameter RAM Memory Map (continued)

Offset[1]	Name	Width	Description
0x22	T_CNT	Hword	Tx internal byte count.
0x24	TTEMP	Word	Tx temporary.

[1]From PIP base address. PIP base = IMMR + 0x3F80 (SMC2)

32.3.1.1 PIP Function Code Register (PFCR)

Figure 32-2 shows the PIP function code register (PFCR).

Bit	0	1	2	3	4	5	6	7
Field	—			**BO**		**AT[1–3]**		
Reset	0000_0000_0000_0000							
R/W	R/W							
Addr	PIP base + 0x04							

Figure 32-2. PIP Function Code Register (PFCR)

Table 32-2 describes the PFCR fields.

Table 32-2. PFCR Field Descriptions

Bits	Name	Description
0–2	—	Reserved and should be 0.
3–4	**BO**	Byte ordering. Set BO to select the required byte ordering for the buffer. If BO is changed on-the-fly, it takes effect at the beginning of the next frame (Ethernet, HDLC, and transparent) or at the beginning of the next BD. See Appendix A, “Byte Ordering.” 00 Reserved 01 Modified little-endian. 1x Big-endian or true little-endian.
5–7	**AT[1–3]**	Address type 1–3. Contains the user-defined function code value used during the SDMA channel memory access. AT0 is always driven high to identify this channel access as a DMA-type access.

32.3.1.2 Status Mask Register (SMASK)

The status mask register (SMASK), shown in Figure 32-3, is important only if the PIP is implementing a Centronics-type transmitter and the CP controls the transfer; see Section 32.9, “Implementing Centronics.” When the CP is handling the DMA transfers, it automatically checks the status lines (from a printer) and masks them against SMASK. Unmasked signals are flagged as errors in the TxBD; see Section 32.5.1, “The PIP Tx Buffer Descriptor (TxBD).”

If the core controls the transmitter, the masking function can be performed in software by reading the individual status signals for errors. When receiving, core software drives the status signals using general-purpose outputs.

Bit	0	1	2	3	4	5	6	7
Field	0				**F**	**PE**	**S**	0
R/W	R/W							
Addr	PIP base + 0x05							

Figure 32-3. Status Mask Register (SMASK)

Table 32-3 describes the SMASK fields.

Table 32-3. SMASK Field Descriptions

Bits	Name	Description
0–3	—	Reserved. Should be 0.
4	**F**	Fault. 0 $\overline{\text{FAULT}}$ status line is ignored. 1 $\overline{\text{FAULT}}$ status line is checked during transmission. If a fault occurs, indication is given in TxBD[F] and a TXE event is generated in the PIPE.
5	**PE**	Paper error. 0 PERROR status line is ignored. 1 PERROR status line is checked during transmission. If a paper error occurs, indication is given in TxBD[PE] and a TXE event is generated in the PIPE.
6	**S**	Select error. 0 SELECT status line is ignored. 1 SELECT status line is checked during transmission. If a select error occurs, indication is given in TxBD[S] and a TXE event is generated in the PIPE.
7	—	Reserved. Should be 0.

32.3.2 PIP Receiver Parameter RAM

The PIP receiver uses the parameter RAM mapping shown in Table 32-4. Certain parameter RAM values must be initialized before the receiver is enabled; others are initialized or written by the CP. Most software does not need access to parameter RAM values after initialization because activity centers around the buffer descriptors.

Table 32-4. PIP Receiver Parameter RAM Memory Map

Offset [1]	Name	Width	Description
0x00	**RBASE**	Hword	PIP Rx BD table base offset from the beginning of dual-port RAM. Initialize RBASE before enabling the channel. RBASE should be divisible by 8.
0x02	—	Hword	Reserved for transmitting.
0x04	**PFCR**	Byte	PIP function code. This value appears on AT[1-3] when the associated SDMA channel accesses memory. Also controls byte ordering for the transfers. See Section 32.3.1.1, “PIP Function Code Register (PFCR).”

Table 32-4. PIP Receiver Parameter RAM Memory Map (continued)

Offset [1]	Name	Width	Description
0x05	—	Byte	Reserved for transmitting.
0x06	**MRBLR**	Hword	Maximum receive buffer length.
0x08	RSTATE	Word	Rx internal state.
0x0C	R_PTR	Word	Rx internal data pointer.
0x10	RBPTR	Hword	RxBD pointer. Points to the current Rx BD being processed or to the next BD to be serviced when idle. After reset or when the end of the Rx BD table is reached, the CP initializes RBPTR to the RBASE value. Most applications should not modify RBPTR, but it can be updated if the receiver is disabled or if no Rx buffer is in use.
0x12	R_CNT	Hword	Rx internal byte count.
0x14	RTEMP	Word	Rx temp.
0x18–0x27	—	—	Reserved for transmitting.
0x28	**MAX_SL**	Hword	Maximum silence period. The PIP controller can be programmed to close the Rx buffer after a period of inactivity determined by MAX_SL.The silence counter decrements every 1,024 system clocks. If the counter reaches zero before new data arrives, the Rx buffer closes. Clearing MAX_SL disables this function.
0x2A	SL_CNT	Hword	Silence counter. Internal-use.
0x2C	**CHARACTER1**	Hword	Control character table. The PIP receiver uses this 8-entry table to support control character recognition. Each entry consists of the control character, a valid bit, and a reject bit. See Section 32.3.2.1, “Control Character Table, RCCM, and RCCR.”
0x2E	**CHARACTER2**	Hword	
⋮	⋮	⋮	
0x3A	**CHARACTER8**	Hword	
0x3C	**RCCM**	Hword	Receive control character mask. See Section 32.3.2.1, “Control Character Table, RCCM, and RCCR.”
0x3E	RCCR	Hword	Receive control character register.See Section 32.3.2.1, “Control Character Table, RCCM, and RCCR.”

[1] Offset from PIP base address. PIP base = IMMR + 0x3F80 (SMC2).

32.3.2.1 Control Character Table, RCCM, and RCCR

The PIP receiver can recognize special control characters used when it functions in a message-oriented environment like Centronics; see Section 32.9, “Implementing Centronics.” Up to eight characters can be defined in the control character table. The user can write an incoming control character to the buffer or reject it. Rejected characters are written to the RCCR in the PIP Rx parameter RAM and reported in the maskable event register PIPE. The rejection method allows the user to handle control characters that are not part of the received message. The PIP receiver uses the structure shown in Figure 32-4 to support control character recognition.

Offset[1]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x2C	**E**	**R**	—						**CHARACTER1**							
0x2E	**E**	**R**	—						**CHARACTER2**							
⋮	⋮	⋮	⋮						⋮							
0x3A	**E**	**R**	—						**CHARACTER8**							
0x3C	**1**	**1**	—						**RCCM**							
0x3E			—						RCCR							

[1] From PIP base address

Figure 32-4. Control Character Table, RCCM, and RCCR

Table 32-5 describes the control character table, RCCM, and RCCR fields.

Table 32-5. Control Character Table, RCCM, and RCCR Descriptions

Offset [1]	Bits	Name	Description
0x2C–0x3A	0	**E**	End of table. In tables with eight control characters, E is always 0. 0 This entry is valid. 1 The entry is not valid and is not used.
	1	**R**	Reject character. 0 The character is not rejected but is written into the Rx buffer, which is then closed. A new buffer is opened if more data is in the message. A maskable interrupt is generated. 1 If this character is recognized it is written to RCCR and not to the Rx buffer. A maskable interrupt is generated. The current Rx buffer is not closed.
	2–7	—	Reserved
	8–15	**CHARACTER*n***	Control character values 1–8. Defines control characters to be compared to the incoming character. For characters smaller than 8 bits, the most significant bits should be zero.
0x3C	0–1	**0b11**	Must be set. Used to mark the end of the control character table in case eight characters are used. Setting these bits ensures correct operation during control character recognition.
	2–7	—	Reserved
	8–15	**RCCM**	Received control character mask. Used to mask the comparison of CHARACTER*n*. RCCM[8–15] correspond to the eight bits of CHARACTER*n* and are decoded as follows. 0 Ignore this bit when comparing the incoming character to CHARACTER*n*. 1 Use this bit when comparing the incoming character to CHARACTER*n*.
0x3E	0–7	—	Reserved
	8–15	RCCR	Received control character register. If the newly arrived character matches and is rejected from the buffer (R = 1), the PIP controller writes the character into the RCCR and generates a maskable interrupt. If the core does not process the interrupt and read RCCR before a new control character arrives, the previous control character is overwritten.

[1] Offset from PIP base address. PIP base = IMMR + 0x3F80 (SMC2).

32.4 The PIP Registers

The PIP registers include one configuration register (PIPC), and an event register (PIPE) with its corresponding mask register (PIPM). A timing parameters register (PTPR) allows the user to program pulsed handshake timings. The port B registers must also be configured for PIP operation. The following subsections describe the PIP registers.

32.4.1 PIP Configuration Register (PIPC)

The PIP configuration (PIPC) register determines all PIP options. Figure 32-5 shows the register format.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	**STR**	—			**SACK**	**CBSY**	**SBSY**	**EBSY**	**TMOD**		**MODL**		**MODH**		**HSC**	**T/R**
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	0xAB2															

Figure 32-5. PIP Configuration Register (PIPC)

This register is affected by $\overline{\text{HRESET}}$ but is not affected by $\overline{\text{SRESET}}$. Table 32-6 describes PIPC fields.

Table 32-6. PIPC Field Descriptions

Bits	Name	Description
0	**STR**	Start transmit. Applies when T/R = 1 (Tx operation). Setting STR causes the CP to poll the TxBD table looking for the next TxBD in which the R-bit is set. Prepare TxBDs and buffers before setting STR. The CP clears STR after one system clock.
1–3	—	Reserved and should be cleared.
4	**SACK**	Set acknowledge. When set, SACK asserts the receiver's $\overline{\text{ACK}}$ output regardless of the receiver state. SACK should be used to implement the IEEE P1284 bidirectional Centronics protocol.
5	**CBSY**	Clear BUSY. When CBSY is set, BUSY is driven low. CBSY is automatically cleared after the PIP negates BUSY. Set EBSY before using SBSY or CBSY. Note that PIPC[T/R] should be cleared (receiving) if SBSY or CBSY are used.
6	**SBSY**	Set BUSY. When SBSY is set, BUSY is driven high. SBSY is automatically cleared after the PIP asserts BUSY. Set EBSY before using SBSY or CBSY. Note that PIPC[T/R] should be cleared (receiving) if SBSY or CBSY are used.
7	**EBSY**	Enable BUSY. The bit definition depends on whether T/R is set to receive or transmit. BUSY is not affected by MODL programming if EBSY = 1. T/R = 0 (Receiving): 0 Disable BUSY signal generation on PB31 for the receiver. 1 Enable the BUSY output on PB31. EBSY takes effect only if BUSY is configured as a PIP output (PBPAR[31] = 0 and PBDIR[31] = 1). T/R = 1 (Transmitting): 0 Ignore the BUSY input on PB31 for the transmitter. 1 Assertion of $\overline{\text{STB}}$ requires negation of BUSY. $\overline{\text{STB}}$ is not asserted until BUSY, input on PB31, is negated. EBSY takes effect only if BUSY is configured as a PIP input (PBPAR[31] = PBDIR[31] = 0).

Table 32-6. PIPC Field Descriptions (continued)

Bits	Name	Description
8–9	**TMOD**	Timing mode. Used to implement a Centronics-type receiver. Valid only when T/R = 0 (Rx operation) and MODH = 11 (pulsed handshake). For the definition of these timing modes, see Section 32.7.2.2, “Pulsed Handshake Timing.” 00 PIP receiver timing mode 0. 01 PIP receiver timing mode 1. 10 PIP receiver timing mode 2. 11 PIP receiver timing mode 3.
10–11	**MODL**	Mode low. Determines the mode of the PIP’s lower 8 signals, PB[24–31], which extend the PIP interface to 16 bits. (If the PIP is 8-bit, program MODL to 0b00.) 00 Port B general-purpose I/O 01 Transparent transfer mode—controlled by the CP. 1x Mode of operation is controlled by MODH. Note that BUSY is not affected by MODL programming if EBSY = 1.
12–13	**MODH**	Mode high. Determines the mode of the PIP upper 10 signals, PB[14–23], which comprise the 8-bit PIP and its control signals. Can be modified when the CP is not transferring data. 00 Port B general-purpose I/O (PIP disabled) 01 Transparent transfer mode—controlled by the CP. 10 Interlocked handshake mode—controlled by the CP or core. 11 Pulsed handshake mode—controlled CP or core.
14	**HSC**	Host control. 0 The CP controls transfers using PIP parameter RAM, buffer descriptors, and SDMA channels. 1 PIP data transfers are controlled by the core.
15	**T/R**	Transmit/receive. Selects transmitter or receiver operation for the PIP. 0 Receive. Data is input to the PIP. 1 Transmit. Data is output from the PIP.

32.4.2 PIP Event Register (PIPE)

The PIP event (PIPE) register is used to generate interrupts and report events recognized by the PIP controller. It shares the same address as the SMC2 event register, which cannot be used at the same time as the PIP. Since PIP is not full duplex, the one PIPE register can report both transmit and receive events concurrently.

When the PIP recognizes an event, it sets the corresponding event bit in the PIPE. PIPE interrupts can be masked in the PIP mask register (PIPM). Writing ones to the PIPE bits clears the events; writing zeros has no effect. All unmasked flags must be cleared before the CP clears internal interrupt requests. Figure 32-6 shows the register format.

Bit	0	1	2	3	4	5	6	7
Field	—			TXE	CCR	BSY	RCH/TCH	RXB/TXB
Reset	0							
R/W	R/W							
Addr	0xA96							

Figure 32-6. PIP Event Register (PIPE)

Table 32-7 describes PIPE fields.

Table 32-7. PIPE Field Descriptions

Bits	Name	Description
0–2	—	Reserved. Should be cleared by writing ones.
3	TXE	Transmit error. Indicates a general transmit error—the source of the specific error can be read in the current buffer descriptor's status and control field; see Section 32.5.1, "The PIP Tx Buffer Descriptor (TxBD)."
4	CCR	Control character received. A control character was received and stored in the received control character register (RCCR) in the PIP parameter RAM.
5	BSY	Busy. Due to a lack of buffers, the PIP did not receive, and therefore discarded, a data byte/half-word.
6	RCH/TCH	Character received/transmitted. Indicates that a data character has been sent or received. Used to generate interrupts to the core if the PIP is configured to be controlled by core software.
7	RXB/TXB	Rx/Tx buffer. Under CP control, indicates that the PIP controller has closed the current buffer due to one of the following events: the transfer byte count reaches zero (for Rx and Tx), a user-defined control character is received and not rejected (R-bit = 0), or the programmable silence period times out.

32.4.3 PIP Mask Register

The PIP mask register (PIPM), whose layout is identical to PIPE, shares address 0xA9A with the SMC2 mask register. Setting a PIPM bit enables the corresponding interrupt; clearing a bit disables the interrupt. PIPM is cleared by reset.

32.4.4 PIP Timing Parameters Register (PTPR)

The PIP timing parameters register (PTPR) holds two timing parameters, TPAR1 and TPAR2, used in the pulsed handshake modes for both sending and receiving. See Section 32.7.2.2, "Pulsed Handshake Timing."

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	**TPAR2**								**TPAR1**							
Reset	0															
R/W	R/W								R/W							
Addr	0xAB6															

Figure 32-7. PIP Timing Parameters Register (PTPR)

This register is affected by $\overline{\text{HRESET}}$ but is not affected by $\overline{\text{SRESET}}$. Table 32-8 describes PTPR fields.

Table 32-8. PTPR Field Descriptions

Bits	Name	Description
0–7	**TPAR1**	Timing parameter 1/2. Defines the number of 860 general system clocks for TPAR1/TPAR2 in transmitter or receiver pulsed handshake mode. The value 0x00 specifies one clock; 0xFF specifies 256 clocks. For a 25-MHz system, the general system clock period is 40 ns.
8–15	**TPAR2**	

32.4.5 The Port B Registers

The PIP uses parallel I/O port B. Figure 32-8 shows the basic operation of port B.

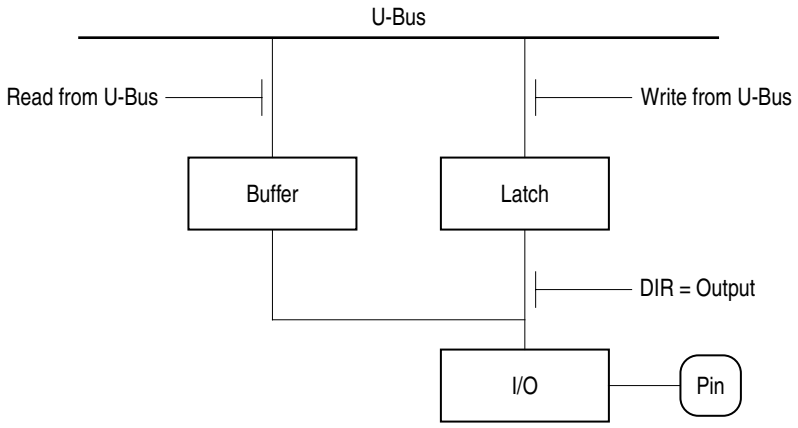


Figure 32-8. Port B General-Purpose I/O

The following describes the configuration of the port B registers for PIP operation; see also Section 33.3.1, “The Port B Registers.”

- Port B assignment register (PBPAR). To use port B for PIP, clear the corresponding PBPAR bits and configure PIPC[MODH] and PIPC[MODL]. See Section 33.3.1.4, “Port B Pin Assignment Register (PBPAR).”
- Port B data direction register (PBDIR). Configures the data signals PBDAT as inputs or outputs. The direction settings for the handshake signals PB14 and PB15 are ignored. See Section 33.3.1.3, “Port B Data Direction Register (PBDIR).”
- Port B data register (PBDAT) register functions as the PIP data register when the PIP is used. Use this register to receive or transmit PIP data when the PIP is controlled by core software. See Section 33.3.1.2, “Port B Data Register (PBDAT).”
- Port B open-drain register. Configures the data signals PBDAT[16–31] as normal or wired-OR. See Section 33.3.1.1, “Port B Open-Drain Register (PBODR).”

32.5 PIP Buffer Descriptors

The CP uses PIP receive and transmit buffer descriptors to manage the specific transfer of each buffer. Each 64-bit buffer descriptor has the following structure:

- The half word at offset + 0 contains status and control bits.
- The half word at offset + 2 contains the data length (in bytes) that is sent or received.
  - For an RxBD, this is the number of octets the CP writes into this RxBD's buffer once the descriptor closes. The controller writes this field after the received data is placed into the associated buffer. Memory allocated for this buffer should be no smaller than MRBLR.
  - For a TxBD, this is the number of octets the CP should transmit from its buffer. However, it is never modified by the CP. This value should be greater than zero. For an 8-bit PIP, this value can be odd or even; for a 16-bit PIP, it must be even.
- The word at offset + 4 points to the beginning of the buffer.
  - For an RxBD, the value must be even and can reside in internal or external memory.
  - For a TxBD, this pointer can be even or odd, unless the port size exceeds 8 bits, for which it must be even. The buffer can reside in internal or external memory.

32.5.1 The PIP Tx Buffer Descriptor (TxBD)

The CP uses buffer descriptors (TxBDs) to confirm buffer transmission and indicate error conditions to the core. Figure 32-9 shows the PIP TxBD.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**R**	—	**W**	**I**	**L**	—	**CM**	—					F	PE	S	—
Offset + 2	Data Length															
Offset + 4	Tx Buffer Pointer															
Offset + 6																

Figure 32-9. PIP Tx Buffer Descriptor (TxBD)

Table 32-9 describes the PIP Tx buffer descriptor status and control field. The data length and buffer pointer are described in Section 32.5, "PIP Buffer Descriptors," above.

Table 32-9. PIP TxBD Status and Control Field Descriptions

Bits	Name	Description
0	**R**	Ready. If PIP tries to transmit a buffer that is not ready, PIPE[TXE] is flagged. 0 The buffer associated with this descriptor is not ready for transmission. This descriptor and its buffer can be updated. The CP clears R after the buffer is sent or an error is encountered. 1 The buffer is ready for sending or is being sent. No fields of this BD can be written while R = 1.
1	—	Reserved and should be cleared.
2	**W**	Wrap (last buffer descriptor in TxBD table). The number of TxBDs in the table is determined only by the W bit and space constraints of the dual-port RAM. 0 Not the last descriptor in the TxBD table. 1 The last BD in the TxBD table. After this BD is processed, the current TxBD pointer wraps to the top of the TxBD table (TBASE).

Table 32-9. PIP TxBD Status and Control Field Descriptions (continued)

Bits	Name	Description
3	**I**	Interrupt. 0 No interrupt is generated after this buffer is serviced. 1 PIPE[TXB] is set when this buffer is serviced by the CP, which can cause an interrupt.
4	**L**	Last. 0 Not the last buffer of the frame. 1 Last buffer of the frame.
5	—	Reserved and should be cleared.
6	**CM**	Continuous mode. 0 Normal operation. 1 The CP does not clear R after this buffer is closed, allowing the associated buffer to be resent when the CP next accesses this BD. However, R is cleared if an error occurs during transmission.
7–11	—	Reserved and should be cleared.
12	F	Fault. 0 The $\overline{\text{FAULT}}$ status line has remained negated during transmission. 1 The $\overline{\text{FAULT}}$ status line has been asserted during transmission.
13	PE	Paper error. 0 The PERROR status line has remained negated during transmission. 1 The PERROR status line has been asserted during transmission.
14	S	Select error. 0 The SELECT status line has remained asserted during transmission. 1 The SELECT status line has been negated during transmission.
15	—	Reserved and should be cleared.

32.5.2 The PIP Rx Buffer Descriptor (RxBD)

Using buffer descriptors, the CP confirms reception or indicates error conditions so the core knows which buffers have been serviced. Figure 32-10 shows the PIP RxBD.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**E**	—	**W**	**I**	C	—	**CM**	SL	—							
Offset + 2	Data Length															
Offset + 4	Rx Buffer Pointer															
Offset + 6																

Figure 32-10. PIP Rx Buffer Descriptor (RxBD)

Table 32-10 describes the PIP RxBD status and control field. The data length and buffer pointer are described in Section 32.5, “PIP Buffer Descriptors,” above.

Table 32-10. PIP RxBD Status and Control Field Descriptions

Bits	Name	Description
0	**E**	Empty. 0 The buffer associated with this descriptor is full or stopped receiving data because an error occurred. The core can read or write any fields of this RxBD. The CP cannot use this BD while E is 0. 1 The buffer associated with this BD is empty or is receiving data. Once E is set, the core should not write any fields of this RxBD.
1	—	Reserved and should be cleared.
2	**W**	Wrap (last buffer descriptor in RxBD table). The number of RxBDs in the table is determined only by the W bit and space constraints of the dual-port RAM. 0 Not the last BD in the RxBD table. 1 The last BD in the RxBD table. After this BD is processed, the current RxBD pointer wraps to the top of the RxBD table (RBASE).
3	**I**	Interrupt. 0 No interrupt is generated after this buffer is filled. 1 PIPE[RXB] is set when the CP fills this buffer, signaling the core to process the buffer. The RXB bit causes an interrupt if not masked.
4	C	Control character 0 This buffer does not contain a control character. 1 This buffer has a user-defined control character as its last byte.
5	—	Reserved and should be cleared.
6	**CM**	Continuous mode. 0 Normal operation. 1 The E bit is not cleared by the CP after this buffer is closed, thus allowing the associated buffer to be automatically overwritten the next time the CP processes this BD.
7	SL	Silence. Indicates that the buffer has closed because the programmable silence period has timed-out.
8–15	—	Reserved and should be cleared.

32.6 PIP CP Commands

The PIP transmit and receive CP commands are the same as the corresponding SMC2 commands (same opcodes and channel number). The PIP transmit commands are described in Table 32-11.

Table 32-11. PIP Transmit CP Commands

Command	Description
STOP TRANSMIT	Disables transmission of frames on the transmit channel. If the PIP controller receives this command during frame transmission, transmission stops and the TBPTR is not advanced to the next BD. No new BD is accessed and no new buffers are sent for this channel. The transmitter idles until RESTART TRANSMIT is issued.

Table 32-11. PIP Transmit CP Commands (continued)

Command	Description
RESTART TRANSMIT	Used to begin or resume sending using the current BD pointed to by TBPTR. When the channel receives this command after PIPC[STR] is set, it starts processing the current BD. The PIP controller expects RESTART TRANSMIT after STOP TRANSMIT is issued, or after a transmitter error occurs.
INIT TX PARAMETERS	Initializes all transmit parameters in the PIP parameter RAM to their reset state and should be issued only when the transmitter is disabled.

The PIP receive commands are described in Table 32-12.

Table 32-12. PIP Receive CP Commands

Command	Description
INIT RX PARAMETERS	Initializes all the receive parameters in the PIP parameter RAM to their reset state and should only be issued when the receiver is disabled.
CLOSE RXBD	Forces the PIP controller to close the current RxBD if it is being used and to use the next BD in the table for subsequent data. No action is taken if the PIP controller is not receiving data.

32.7 Handshaking I/O Modes

In either handshaking I/O mode, interlocked or pulsed, the PIP can be configured as a transmitter or receiver and either the CP or the core can control communications. For CP control, BD and parameter RAM initialization is required; data is stored in the buffers using the SDMA channels dedicated to SMC2. For core control, software interrupt routines read and write to the PIP data register (PBDAT).

When the PIP transmits, STBO (PB15) is the $\overline{\text{STB}}$ handshake control signal and STBI (PB14) is the $\overline{\text{ACK}}$ input. When the PIP receives, it generates $\overline{\text{ACK}}$ on STBO in response to $\overline{\text{STB}}$ on STBI. Note that the PIP controller overrides bits 15 and 14 in the port B data direction register (PBDIR) and PBDAT corresponding to STBO and STBI. (The open-drain register PBODR does not apply to PB15 and PB14.)

The following subsections describe interlocked and pulsed handshake modes.

32.7.1 Interlocked Handshake Mode

The interlocked handshake mode provides a fast connection between MPC860s and can be used for P1284-protocol advanced byte mode transfers. To connect MPC860s using this interface, connect STBO from one 860 to the STBI of the other and connect the appropriate data signals (either PB[23–16] or PB[31–16]).

When the PIP is transmitting, the CP loads data into the output latch when it receives a request from the core to begin transfers. Once data is loaded, $\overline{\text{STB}}$ is asserted after a programmable setup time. When $\overline{\text{ACK}}$ is sampled as low, data is sent, and $\overline{\text{STB}}$ is negated. $\overline{\text{STB}}$ remains negated until new data is loaded into the output latch and $\overline{\text{ACK}}$ is negated.

When the PIP is receiving, input data is latched when $\overline{STB}$ is sampled as low. $\overline{ACK}$ is asserted and then negated after the data is removed from the input latch. Figure 32-11 shows the handshake timing of the interlocked mode.

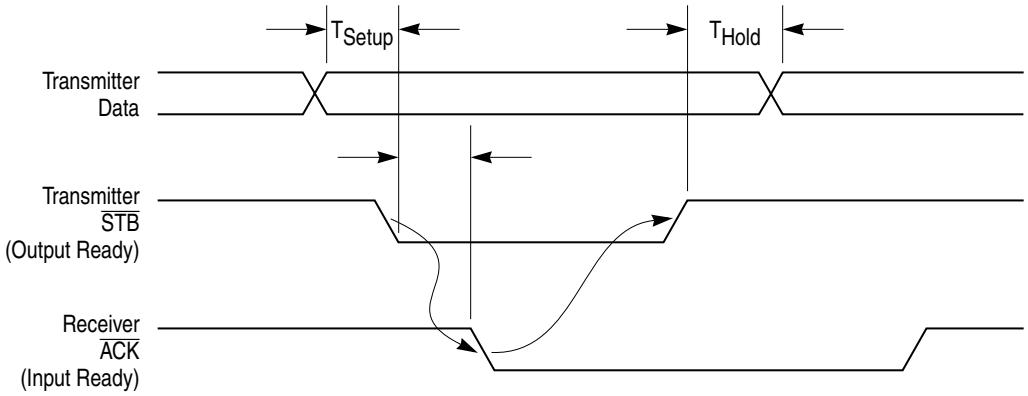


Figure 32-11. Interlocked Handshake Mode Timing

32.7.2 Pulsed Handshake Mode

The pulsed handshake mode, shown in Figure 32-12, supports a Centronics-compatible interface.

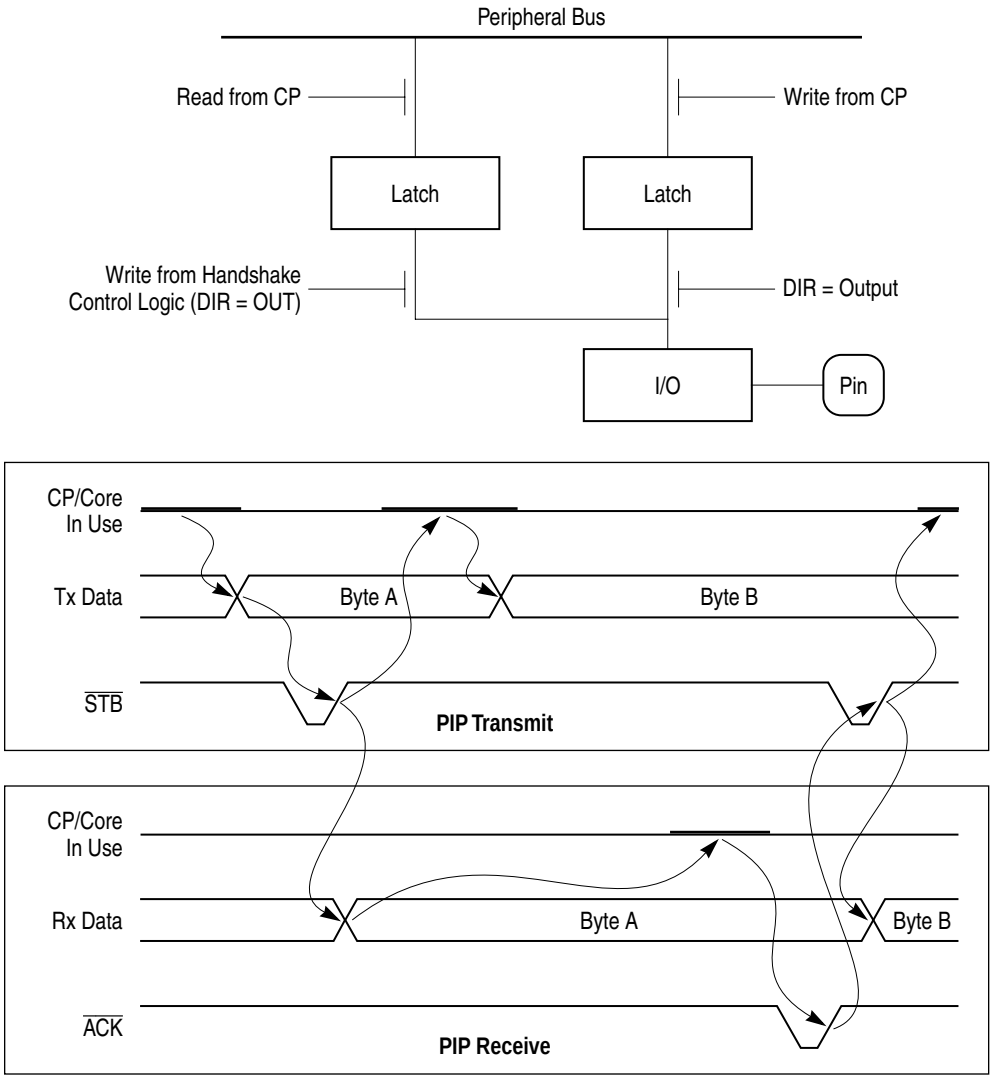


Figure 32-12. Pulsed Handshake Full Cycle

- When sending, the PIP generates $\overline{\text{STB}}$ when data is ready in the PIP output latch and the previous transfer is acknowledged. The setup time and the pulse width of $\overline{\text{STB}}$ are programmable.
- When receiving, the PIP uses $\overline{\text{STB}}$ to latch input data and $\overline{\text{ACK}}$ to acknowledge the transfer. The timing of $\overline{\text{ACK}}$ is also programmable.

The core configures the PIP to implement a Centronics protocol by programming the PIP configuration (PIPC) register. When the PIP is under CP control, timing attributes are set in PTPR. Transmit and receive errors are reported through BDs. For information about supporting a Centronics interface, see Section 32.9, “Implementing Centronics.”

32.7.2.1 The BUSY Signal

In pulsed handshake mode, the PIP receiver can generate an additional BUSY handshake signal that is useful when implementing a Centronics receiver interface. The BUSY output is used to indicate a transfer in progress; the PIP receiver asserts BUSY as soon as data is latched into the PIP data register. Figure 32-13 shows the pulsed handshake timing including a BUSY signal.

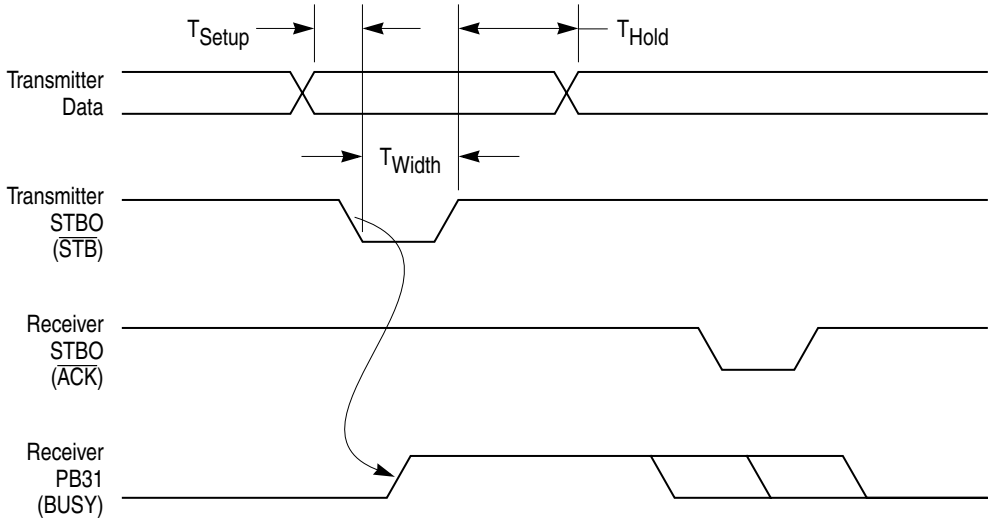


Figure 32-13. Pulsed Handshake BUSY Signal

The timing of BUSY negation can be programmed relative to $\overline{\text{ACK}}$; see Section 32.7.2.2, “Pulsed Handshake Timing.” Core software can also control the assertion and negation of BUSY via PIPC; see Section 32.4.1, “PIP Configuration Register (PIPC).”

BUSY is multiplexed onto PB31. It can be used only with the 8-bit PIP interface (not the 16-bit interface). A PIP transmitter can be configured to ignore BUSY or suspend assertion of the $\overline{\text{STB}}$ output until the receiver BUSY signal is negated.

32.7.2.2 Pulsed Handshake Timing

When the PIP is under CP control, the pulsed-handshake timing parameters are governed by PTPR[TPAR*n*] fields, which define an interval from 1 to 256 system clocks; see Section 32.4.4, “PIP Timing Parameters Register (PTPR).”

Figure 32-14 shows how the timing parameter TPAR1 governs the setup time and TPAR2 defines the pulse width of $\overline{\text{STB}}$ of a PIP transmitter using pulsed handshake mode timing.

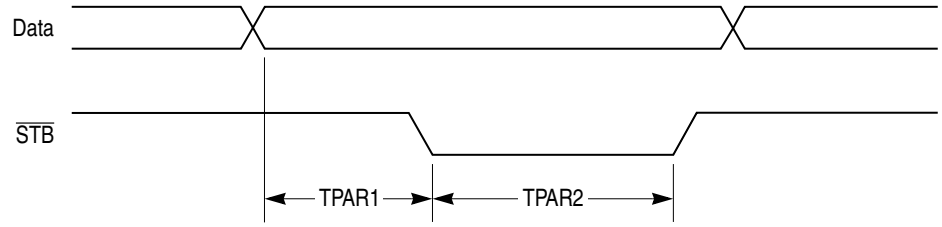


Figure 32-14. PIP Transmitter Timing Diagram

A PIP receiver in pulsed handshake mode has four options for determining the relative timing of BUSY to ACK. Figure 32-15 through Figure 32-18 show how the definitions of TPAR1 and TPAR2 vary for each receiver mode. The receiver mode is selected in PIPC[TMOD]; see Table 32-6.

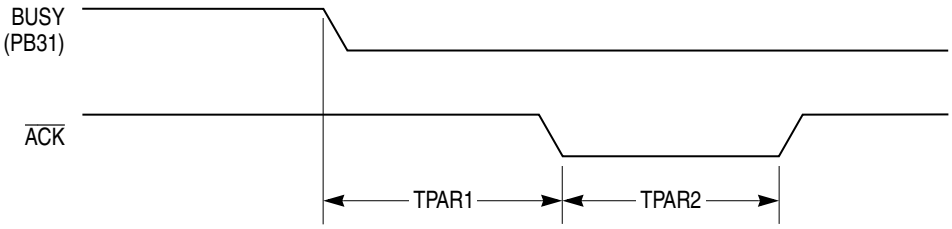


Figure 32-15. PIP Receiver Timing—Mode 0

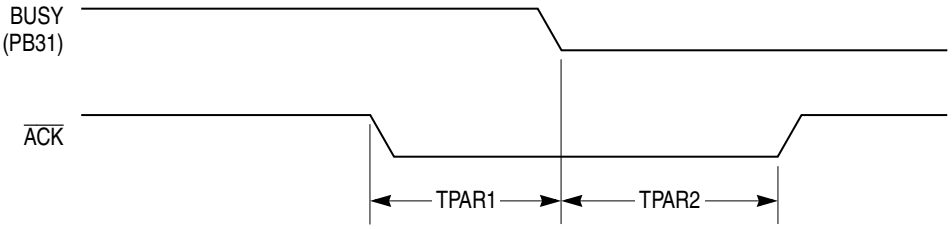


Figure 32-16. PIP Receiver Timing—Mode 1

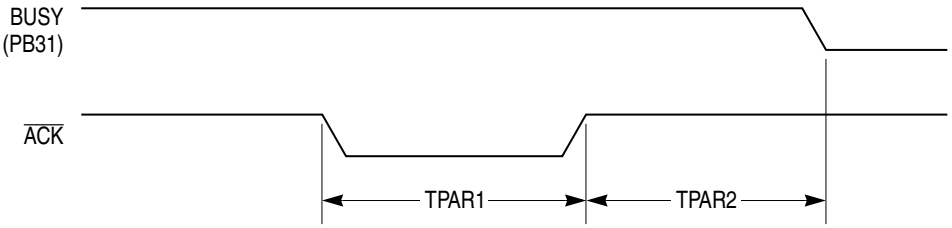


Figure 32-17. PIP Receiver Timing—Mode 2

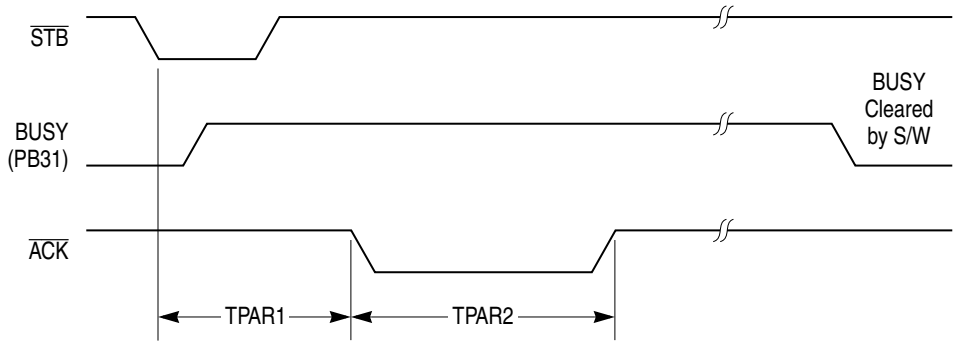


Figure 32-18. PIP Receiver Timing—Mode 3

32.8 Transparent Transfers

The timing of transparent transfers, shown in Figure 32-19, is controlled by the timing of the strobe signal STBI (PB14). Transparent transfers must be controlled by the CP, which requires BD and parameter RAM initialization. The CP moves data in and out of buffers using the virtual SDMA channels dedicated to SMC2. The falling edge of STBI generates the request to the CP causing it to send or receive data. Signal direction is controlled by the port B data direction register (PBDIR[14]).

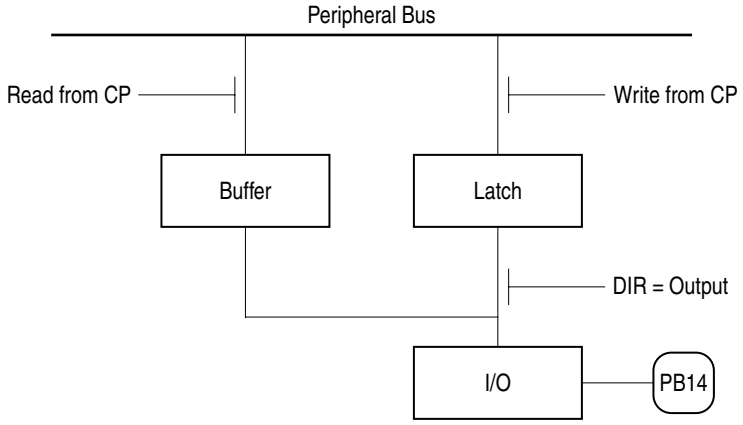


Figure 32-19. PIP Transparent Transfers

32.9 Implementing Centronics

The PIP can implement a Centronics-compatible interface for both sending and receiving. The Centronics protocol is a parallel peripheral interface for communicating between a host computer and a printer. To implement Centronics, the PIP uses an 8-bit data bus, two handshake signals that control the data exchange, and signals that reflect the peripheral device status.

Traditionally, Centronics transfers have been one-way from the host to a peripheral device, but new standards like IEEE P1284 allow bidirectional transfers. With software to allow switching between receive and transmit modes, the PIP can support bidirectional transfers, but does not fully comply with the full-duplex P1284 standard interface.

The following is a list of the PIP controller's Centronics-compatible features:

- Superset of the Centronics standard
- Supports Centronics-type transmitter and receiver operating modes
- Supports bidirectional Centronics
- Message-oriented data structure flexibility
- Flexible receive control character comparison
- Flexible timing modes with programmable timing parameters

Figure 32-20 shows the signals needed to implement a standard Centronics interface.

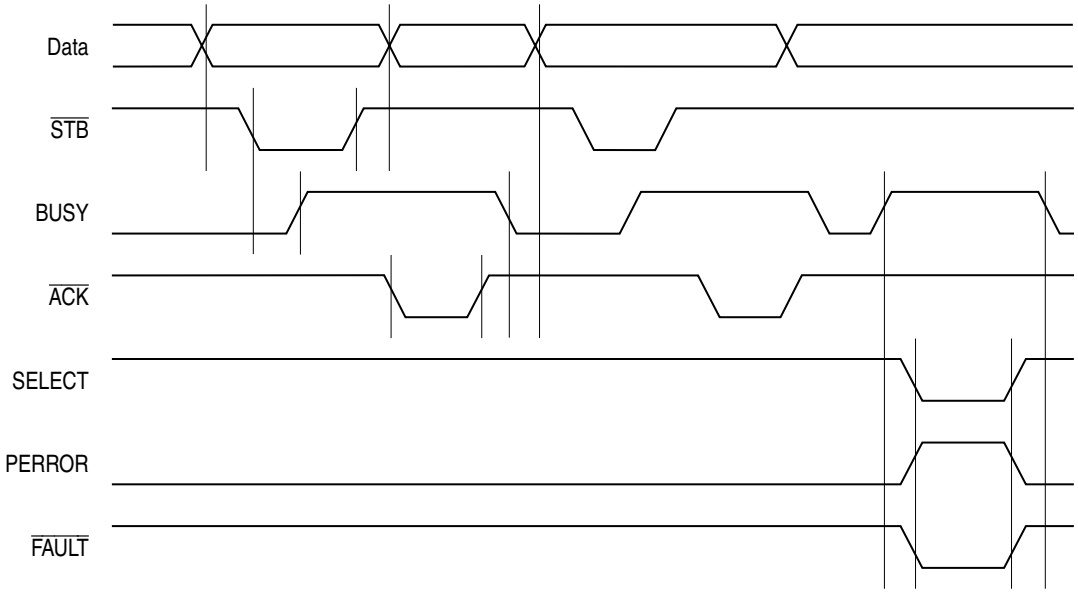


Figure 32-20. The PIP Centronics Interface Signals

The following subsections describe the PIP configured as a Centronics interface.

32.9.1 PIP as a Centronics Transmitter

Once the TxBDs are prepared and PIPC[STR] is set, the PIP processes the next ready BD in the TxBD table. When configured for a Centronics interface, the PIP transmitter fetches data from memory and starts sending to the printer. Assuming the corresponding status mask bits are set in SMASK, the PIP transmitter checks the printer status lines (SELECT, PERROR and $\overline{\text{FAULT}}$) for Tx errors before each transfer. Configure PB30, PB29, and PB28

as general-purpose inputs and connect them to SELECT, PERROR, and $\overline{\text{FAULT}}$, respectively.

For each transfer, the PIP drives the data on the Centronics interface data lines and generates a strobe pulse, assuming the previous data has been acknowledged and the minimum setup time requirement is met. Strobe pulse width and setup time parameters are set in the PIP timing parameters register (PTPR). Note that one data frame can span several buffers with a maskable interrupt generated after each BD is processed.

Figure 32-21 shows the PIP configured as a Centronics transmitter.

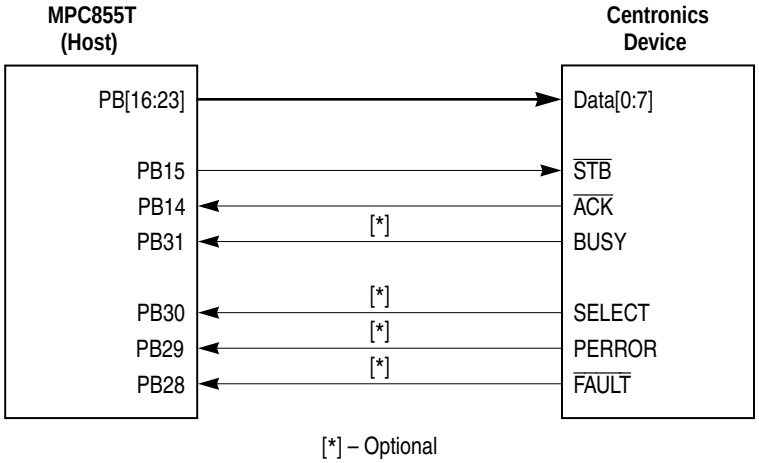


Figure 32-21. PIP as a Centronics Transmitter

32.9.1.1 Centronics Tx Errors and the PIPE

The Centronics transmission errors are described in Table 32-13.

Table 32-13. Centronics Tx Errors

Error	Description
BD Not Ready	The current BD to be processed is not ready. PIPE[TXE] is flagged. The channel continues sending after S/W prepares the BD and sets PIPC[STR].
Printer Off-Line	The printer is off-line. TxBD[S] and PIPE[TXE] are flagged. The channel resumes sending after RESTART TRANSMIT. Note that SMASK[S] must be set to sense this printer status line when the CP is controlling the transfer.
Printer Fault	The printer has a fault condition. TxBD[F] and PIPE[TXE] are flagged. The channel resumes sending after RESTART TRANSMIT. Note that SMASK[F] must be set to sense this printer status line when the CP is controlling the transfer.
Paper Error	The printer has an error in its paper path. TxBD[PE] and PIPE[TXE] are flagged. The channel resumes sending after RESTART TRANSMIT. Note that SMASK[PE] must be set to sense this printer status line when the CP is controlling the transfer.

The relevant PIPE flags during Centronics transmission are TXE, TCH, and TXB; see Section 32.4.2, “PIP Event Register (PIPE).” For core-controlled transmissions, only the character-based TCH interrupt applies.

32.9.2 PIP as a Centronics Receiver

If the current BD in the RxBD table is empty and a character is received from the Centronics interface, the PIP receiver first compares the character against the user-defined control character table. If no match is found, the character is written to the buffer. If a match is found, the control character is either written to the Rx buffer or rejected, depending on the reject bit in the control character table. If rejected, the character is written to the received control character register (RCCR) in the PIP Rx parameter RAM and a maskable interrupt is generated when the BD finishes processing. Note that a single received data frame can span several buffers.

For each transfer, the PIP controller generates $\overline{\text{ACK}}$ and BUSY handshake signals on the Centronics interface. The $\overline{\text{ACK}}$ pulse width and the timing of BUSY with respect to $\overline{\text{ACK}}$ are determined by the timing parameter register PTPR.

Figure 32-22 shows the PIP configured as a Centronics receiver. The SELECT, PERROR, and $\overline{\text{FAULT}}$ signals shown are not automatically generated; they are controlled by software and driven on general-purpose outputs.

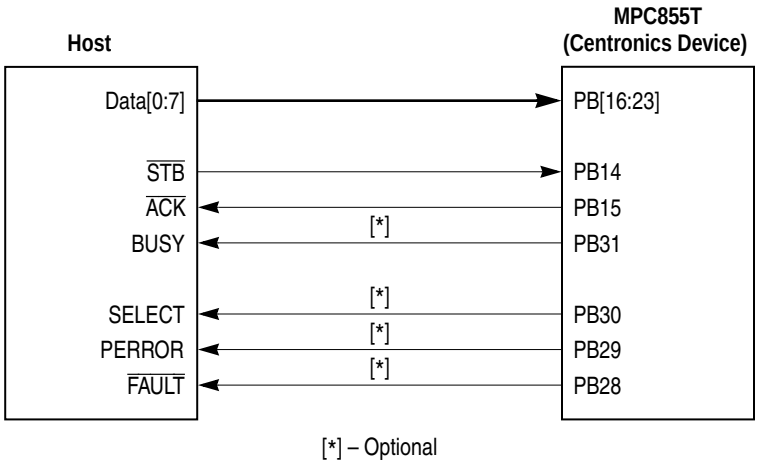


Figure 32-22. PIP as a Centronics Receiver

32.9.2.1 Centronics Rx Errors and the PIPE

The Centronics receiving error is described in Table 32-14.

Table 32-14. Centronics Rx Error

Error	Description
BD Busy	The current BD to be processed is not empty. PIPE[BSY] is flagged. The channel resumes receiving after user software prepares the BD.

The relevant PIPE event bits for Centronics receiving are CCR, BSY, RCH, and RXB; see Section 32.4.2, “PIP Event Register (PIPE).” For core-controlled receiving, only the character-based RCH interrupt applies.

Chapter 33
Parallel I/O Ports

The CPM supports four general-purpose I/O ports—A, B, C, and D. Each signal in the I/O ports can be configured as a general-purpose I/O signal or as a signal dedicated to supporting communications devices, such as SMCs and SCC1.

- Port A is shared with the RXD and TXD signals of SCC1, the bank of clock signals, and some time-division multiplexed (TDM) signals.
- Port B is shared with the parallel interface port (PIP) and other functions such as TDM, IDMA, SMC, SPI, UTOPIA, and I^2C signals.
- Port C is shared with the $\overline{\text{RTS}}$, $\overline{\text{CTS}}$, and $\overline{\text{CD}}$ signals of SCC1 as well as some TDM signals. However, port C is unique in that its signals can generate interrupts to the CPM interrupt controller (CPIC).
- Port D is for general-purpose I/O shared with SAR-specific, TDM signals, and Ethernet CAM-support signals.

The read/write port signals can be configured as inputs or outputs with a latch for data output. They can be configured to be either general-purpose I/O or dedicated peripheral signals. Regardless of the programmed function, the I/O signals' state can always be read from their data registers (P*x*DAT).

Ports A and B have signals that can be configured as open-drain. Open-drain signals drive a zero voltage, but they three-state when driving a high voltage. Note that none of the port signals have internal pull-up resistors.

To support flexible configuration of the CPM, many dedicated peripheral functions are multiplexed onto ports A, B, C, and D. Functions are grouped to maximize the signals' usefulness to the greatest number of MPC855T applications. To understand signal assignments described in this chapter, it helps to understand each CPM peripheral.

33.1 Features

The following lists the main features of the parallel I/O ports:

- Port A is 16 bits
- Port B is 18 bits. Port B is shared with the PIP, which is described in Chapter 32, “Parallel Interface Port (PIP).”
- Port C is 12 bits
- Port D is 13 bits
- All ports are bidirectional
- All ports are three-stated at hardware reset
- All ports have alternate on-chip peripheral functions and all signal values can be read while the signal is connected to an on-chip peripheral
- Ports A and B have open-drain capability
- Port C has 12 interrupt input signals

33.2 Port A

Port A signals are configured as follows in the port A pin assignment register (PAPAR):

- General-purpose I/O signal (the corresponding PAPAR[DD*n*] = 0)
- Dedicated on-chip peripheral signal (PAPAR[DD*n*] = 1)

PAPAR and the port A data direction register (PADIR) are cleared at reset, thus configuring all port A signals as general-purpose input signals. Table 33-1 shows defaults for port A signal options.

Table 33-1. Port A Pin Assignment

Signal	Pin Function			
	PAPAR[DD*n*] = 0 (General I/O) [1]	PAPAR[DD*n*] = 1		Input to On-Chip Peripherals (Default)
		PADIR[DR*n*] = 0	PADIR[DR*n*] = 1	
PA15	PORT A15	RXD1	—	GND
PA14	PORT A14	TXD1	—	—
PA13	PORT A13	—	—	GND
PA12	PORT A12	—	—	—
PA11	PORT A11	—	—	Undefined
PA10	PORT A10	—	—	GND
PA9	PORT A9	—	L1TXDA	Undefined
PA8	PORT A8	—	L1RXDA	L1RXDA = GND
PA7	PORT A7	CLK1/TIN1/L1RCLKA [2]	BRGO1	CLK1/TIN1/L1RCLKA = BRGO1
PA6	PORT A6	CLK2	$\overline{\text{TOUT1}}$	CLK2 = GND

Table 33-1. Port A Pin Assignment (continued)

Signal	Pin Function			
	PAPAR[DD*n*] = 0 (General I/O) [1]	PAPAR[DD*n*] = 1		Input to On-Chip Peripherals (Default)
		PADIR[DR*n*] = 0	PADIR[DR*n*] = 1	
PA5	PORT A5	CLK3/TIN2/L1TCLKA[2]	BRGO2	CLK3/TIN2/L1TCLKA = BRGO2
PA4	PORT A4	CLK4	$\overline{\text{TOUT2}}$	CLK4 = CLK8
PA3	PORT A3	CLK5/TIN3[2]	BRGO3	CLK5/TIN3 = BRGO3
PA2	PORT A2	CLK6	$\overline{\text{TOUT3}}$	CLK6 = GND
PA1	PORT A1	CLK7/TIN4[2]	BRGO4	CLK7/TIN4 = BRGO4
PA0	PORT A0	CLK8	$\overline{\text{TOUT4}}$	CLK8 = GND

[1] Clearing the corresponding PADIR bit makes the signal an input; setting PADIR makes it an output.
[2] Multi-function peripheral input signals, such as CLK1/TIN1/L1RCLKA, can perform multiple functions simultaneously. (That is, a clock supplied at PA7 can be used for both CLK1 and TIN1.)

Port A signals selected for general-purpose I/O can be accessed through the port A data register (PADAT). Data written to PADAT is stored in an output latch. For port A outputs, the latch data is gated onto the signal. When PADAT is read, the signal itself is read. For inputs, data written to PADAT is also stored in the output latch but cannot reach the port signal, so when PADAT is read, the signal's state is read. If an input to a peripheral is not supplied from a signal, the default value listed in Table 33-1 is supplied.

33.2.1 Port A Registers

Port A has four memory-mapped control registers, described in the following sections.

33.2.1.1 Port A Open-Drain Register (PAODR)

The port A open-drain register (PAODR), shown in Figure 33-1, determines which port signals with serial channel output capability are configured in a normal or wired-OR configuration. Setting the PAODR bits configure the signals for open-drain operation.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—								OD8	OD9	OD10	OD11	OD12	—	OD14	—
Reset	0															
R/W	R/W															
Addr	0x954															

Figure 33-1. Port A Open-Drain Register (PAODR)

Table 33-2 describes PAODR bits.

Table 33-2. PAODR Bit Descriptions

Bits	Name	Description
0–7, 13, 15	—	Reserved, always reads as 0.
8–12, 14	OD*n*	Tells how the corresponding port A signal is interpreted. 0 The signal is actively driven as an output. 1 The signal is an open-drain driver. Outputs are actively driven low. Otherwise, it is three-stated.

33.2.1.2 Port A Data Register (PADAT)

Reading the port A data (PADAT) register returns the value of the signal, regardless of whether the signal is an input or output. Comparing written data with the data on the signal can detect output conflicts. A write to a PADAT bit is latched; if the bit is configured as an output, the value latched for that bit is driven onto its respective signal. PADAT can be read or written at any time, is not initialized, and is undefined at reset.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	D0	D1	D2	D3	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	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	0x956															

Figure 33-2. Port A Data Register (PADAT)

Table 33-3 describes PADAT bits.

Table 33-3. PADAT Bit Descriptions

Bits	Name	Description
0–15	D*n*	Contains the data on the corresponding signal.

33.2.1.3 Port A Data Direction Register (PADIR)

Port A data direction register (PADIR) bits configure port A signals as general-purpose inputs or outputs. If a signal is not programmed for general-purpose I/O, PADIR selects the peripheral function to be performed.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	DR0	DR1	DR2	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	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	0x950															

Figure 33-3. Port A Data Direction Register (PADIR)

Table 33-4 describes PADIR bits.

Table 33-4. PADIR Bit Descriptions

Bits	Name	Description
0–15	DR*n*	Port A data direction. Configures port A signals as inputs or outputs when functioning as general-purpose I/O; otherwise, used to select the peripheral function. 0 Select the signal for general-purpose input, or select peripheral function 0. 1 Select the signal for general-purpose output, or select peripheral function 1.

33.2.1.4 Port A Pin Assignment Register (PAPAR)

The port A pin assignment register (PAPAR) configures signals as general-purpose I/O or dedicated for use with a peripheral.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	DD0	DD1	DD2	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	0x952															

Figure 33-4. Port A Pin Assignment Register (PAPAR)

Table 33-5 describes PAPAR bits.

Table 33-5. PAPAR Bit Descriptions

Bits	Name	Description
0–15	DD*n*	Configures a signal for general-purpose I/O or for dedicated peripheral function 0 General-purpose I/O. The peripheral functions of the signal are not used. 1 Dedicated peripheral function. The signal is used by the internal module. The on-chip peripheral function to which it is dedicated can be determined by other bits.

33.2.2 Port A Configuration Examples

This section describes the configuration for several PA signals, which are as follows:

- PA15 can be configured as a general-purpose I/O signal but not as an open-drain signal. It can also be RXD1 for SCC1 in NMSI (nonmultiplexed serial interface) mode. If it is configured as a general-purpose I/O signal, the RXD1 input is internally grounded. If SCC1 is connected to a TDM or is not used, PA15 can be used for general-purpose I/O. See Section 33.2.3, “Port A Functional Block Diagrams.”
- PA14 can be configured as a general-purpose I/O signal, either open-drain or not. See Section 33.2.3, “Port A Functional Block Diagrams.”
  - If PA14 is configured as a general-purpose I/O signal, the TXD1 output is not connected externally. If SCC1 is connected to a TDM or is not used, PA14 can be used for general-purpose I/O.
  - In NMSI mode, if TXD1 is an output on PA14 and PAODR[OD14] = 1, TXD1

is an open-drain output from SCC1.

- PA11 can be configured as a general-purpose I/O and an open-drain signal.PA7 can be configured as a general-purpose I/O signal but not an open-drain signal.
  - If PADIR[DR7] = 0, PA7 can also be CLK1, TIN1, L1RCLKA, or all three. The connections are made separately in the serial interface and timer mode registers.
  - If PADIR[DR7] = 1, PA7 can also be BRGO1. If PA7 is a general-purpose I/O signal, the input to the on-chip peripheral is connected internally to BRG01. Chapter 20, “Serial Interface,” describes CLK1 and L1RCLKA.
- PA4 can be configured as a general-purpose I/O signal but not an open-drain signal.
  - If PADIR[DR4] = 0, PA4 can be CLK4. If DR4 = 1, PA4 can be $\overline{\text{TOUT2}}$.
  - If PA4 is a general-purpose I/O signal, the on-chip CLK4 function is provided via CLK8 (default input). This is useful because CLK signals cannot always be routed to all serial channels. See Chapter 20, “Serial Interface.”

33.2.3 Port A Functional Block Diagrams

Using PA15 as an example, Figure 33-5 shows the functional block diagram for all port A signals without open-drain capability.

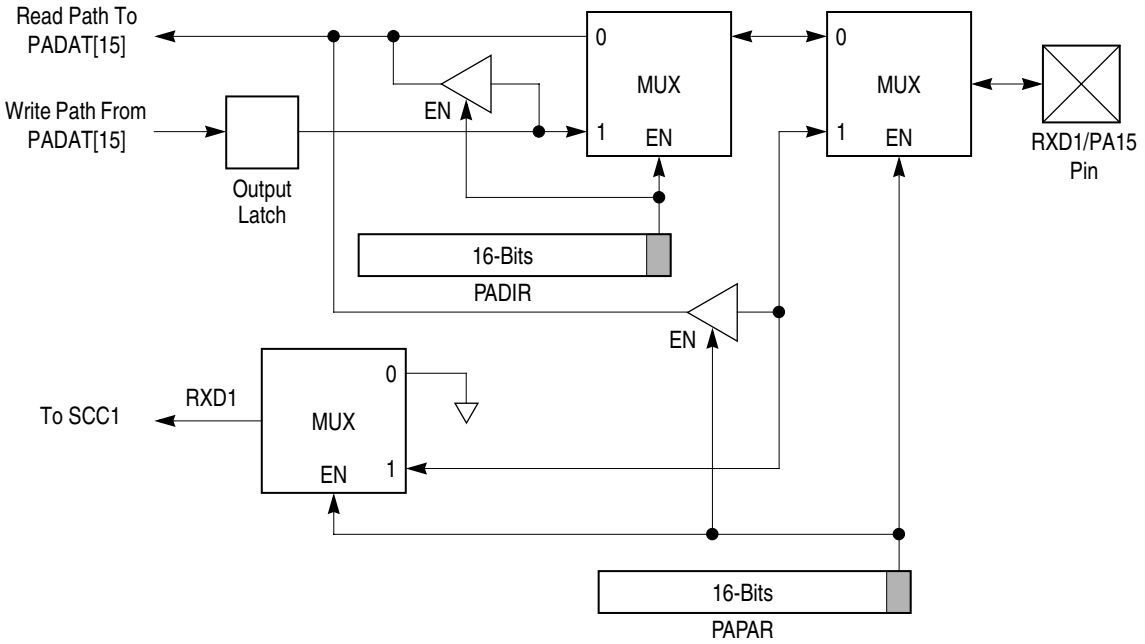


Figure 33-5. Block Diagram for PA15 (True for all Non-Open-Drain Port Signals)

Using PA14 as an example, Figure 33-6 shows the functional block diagram for all port A signals with open-drain capability.

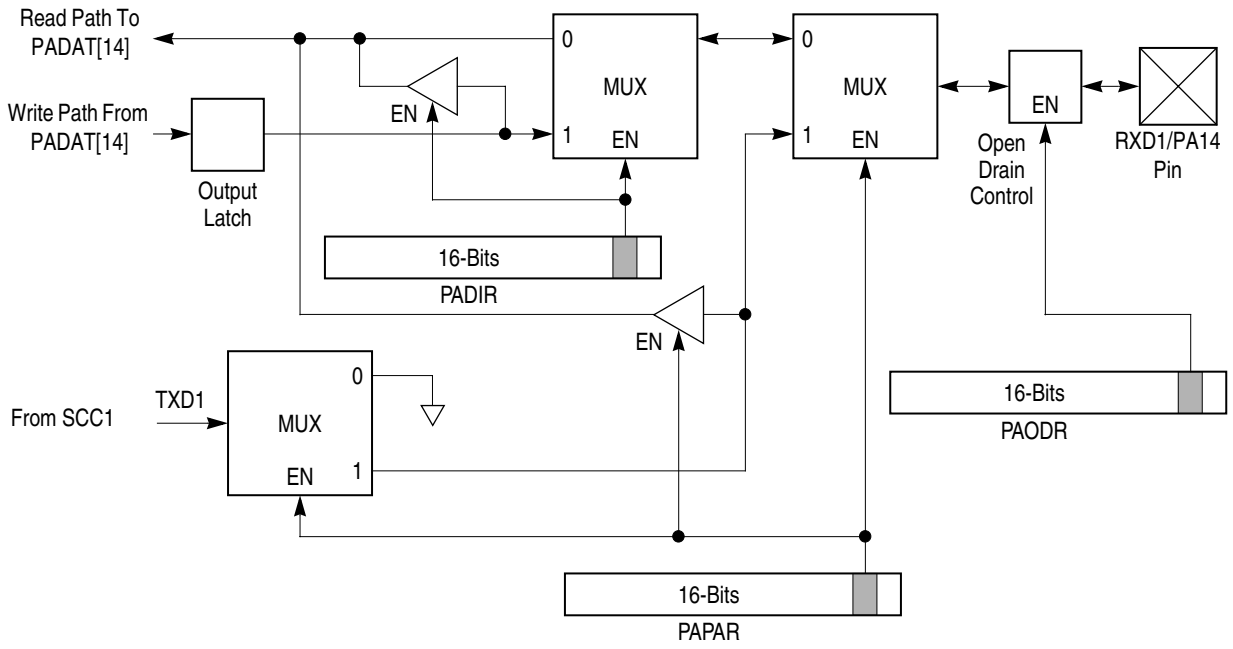


Figure 33-6. Block Diagram for PA14 (True for all Open-Drain Port Signals)

33.3 Port B

All port B signals can be open-drain. They are configured independently as general-purpose I/O signals if the corresponding bit in the PBPAR is cleared and they are configured as dedicated on-chip peripheral signals if the corresponding PBPAR bit is set. When configured as a general-purpose I/O signal, the signal direction of that signal is determined by the corresponding control bit in the PBDIR. The port I/O signal is configured as an input if the corresponding PBDIR bit is cleared and it is configured as an output if the corresponding PBDIR bit is set. All PBPAR bits and PBDIR bits are cleared by hardware reset, thus configuring all port B signals as general-purpose inputs. Table 33-6 describes port B signal options. Port B is shared with the PIP, which is described in Chapter 32, “Parallel Interface Port (PIP).”

If a port B signal is selected as a general-purpose I/O signal, it can be accessed through the PBDAT where data is stored in an output latch. If a port B signal is configured as an output, the output latch data is gated onto the port signal. When PBDAT is read, the port signal itself is read.

All port B signals can have multiple configurations, which include on-chip peripheral functions for SPI, I^2C, SMCs, and the TDM. Port B is also multiplexed with the PIP, which can implement fast parallel interfaces. For a description of the dedicated PIP signal functions, see Chapter 32, “Parallel Interface Port (PIP).”

PB[26–28], and PB[15] are special in that their on-chip peripheral functions (BRGO*x*) are also available in port A. This allows an alternate way to output BRG signals if other

functions are used. PB[16–19] are special in that their on-chip peripheral functions ($\overline{\text{RTS}}x$ and L1ST*x*) are available in port C providing an alternate location to output these signals if other functions on port C are used. (The STBI and STBO signals (PB14 and PB15) used by the PIP are not listed in Table 33-6. Section 32.7.1, “Interlocked Handshake Mode,” gives instructions for enabling them.)

Table 33-6. Port B Pin Assignment

Signal	Signal Function			
	PBPAR[DD*n*] = 0	PBPAR[DD*n*] = 1		Input to On-chip Peripherals (Default)
		PBDIR[DR*n*] = 0	PBDIR[DR*n*] = 1	
PB31	Port B31	$\overline{\text{REJECT1}}$	$\overline{\text{SPISEL}}$	V_{DD}
PB30	Port B30	—	SPICLK	SPICLK = GND
PB29	Port B29	—	SPIMOSI	SPIMOSI = V_{DD}
PB28	Port B28	BRGO4	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	—	SMRXD1 = GND
PB23	Port B23	$\overline{\text{SMSYN1}}$	$\overline{\text{SDACK1}}$	$\overline{\text{SMSYN1}}$ = GND
PB22	Port B22	$\overline{\text{SMSYN2}}$	$\overline{\text{SDACK2}}$	$\overline{\text{SMSYN2}}$ = GND
PB21	Port B21/PHYSEL[1] [1]	SMTXD2	—	—
PB20	Port B20/PHYSEL[0] [2]	SMRXD2	L1CLKOA	SMRXD2 = GND
PB19	Port B19	L1ST1	$\overline{\text{RTS1}}$	—
PB18	Port B18	L1ST2	$\overline{\text{RTS1}}$	—
PB17	Port B17/PHYREQ[1]	L1ST3	—	—
PB16	Port B16/PHYREQ[0]	L1ST4	$\overline{\text{L1RQa}}$	—
PB15	Port B15	TxClav	BRGO3	—
PB14	Port B14	—	$\overline{\text{RSTRT1}}$	—

1 PBDIR[DRn]=1

2 PBDIR[DRn]=1

33.3.1 The Port B Registers

The four port B control registers determine whether a signal is open-drain, input or output, and general-purpose or dedicated to a peripheral.

33.3.1.1 Port B Open-Drain Register (PBODR)

The port B open-drain register (PBODR) indicates when the port signals are configured in a normal or wired-OR configuration. Bits 14 and 15 of PBODR are not implemented. PBODR is cleared by system reset.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Reset	0000_0000_0000_0000															
R/W	—															
Addr	0xAC0															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	OD16	OD17	OD18	OD19	0D20	0D21	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	0xAC2															

Figure 33-7. Port B Open-Drain Register (PBODR)

Table 33-7 describes PBODR bits.

Table 33-7. PBODR Bit Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	OD*n*	Port B open-drain configuration. 0 The I/O signal is actively driven as an output. 1 The I/O signal is an open-drain driver. As an output, the signal is actively driven low. Otherwise, it is three-stated. Note that SMTXD1 cannot be configured as an open-drain driver, regardless of PBODR[OD25].

33.3.1.2 Port B Data Register (PBDAT)

Reading the port B data register (PBDAT) returns data to the signal, regardless of whether it is an input or output. This allows output conflicts to be found on the signal by comparing the written data with the data on the signal. Data written to PBDAT is latched; if the corresponding PBDIR bit is configured as an output, the latched value is driven onto its respective signal. PBDAT can be read or written at any time and is not initialized.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Reset	Undefined															
Addr	0xAC4															
Bits	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

Figure 33-8. Port B Data Register (PBDAT)

Reset	Undefined															
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	0xAC6															

Figure 33-8. Port B Data Register (PBDAT)

Table 33-8 describes PBDAT bits.

Table 33-8. PBDAT Bit Descriptions

Bits	Name	Description
0–13	—	Reserved
14–31	D*n*	Contains the data on the corresponding signal.

33.3.1.3 Port B Data Direction Register (PBDIR)

Port B data direction register (PBDIR) bits configure port B signals as general-purpose inputs or outputs. If a signal is not programmed for general-purpose I/O, PBDIR selects the peripheral function to be performed.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—														DR14	DR15
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0	0
R/W	—														R/W	R/W
Addr	0xAB8															
Bits	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	0xABA															

Figure 33-9. Port B Data Direction Register (PBDIR)

Table 33-9 describes PBDIR bits.

Table 33-9. PBDIR Bit Descriptions

Bits	Name	Description
0–13	—	Reserved
14–31	DR*n*	Port B data direction. Configures port B signals as inputs or outputs when functioning as general-purpose I/O; otherwise, used to select the peripheral function. 0 Select the signal for general-purpose input, or select peripheral function 0. 1 Select the signal for general-purpose output, or select peripheral function 1. DR14 and DR15 are ignored when port B is used by the PIP controller.

33.3.1.4 Port B Pin Assignment Register (PBPAR)

The port B pin assignment register (PBPAR) configures signals as general-purpose I/O or dedicated for use with a peripheral.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—														DD14	DD15
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0	0
R/W	—														R/W	R/W
Addr	0xABC															
Bits	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
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	0xABE															

Figure 33-10. Port B Pin Assignment Register (PBPAR)

Table 33-10 describes PBPAR bits.

Table 33-10. PBPAR Bit Descriptions

Bits	Name	Description
0–13	—	Reserved
14–31	DD*n*	Port assignment. Determines whether a signal is configured for general-purpose I/O or dedicated peripheral function. 0 General-purpose I/O. The peripheral functions of the signal are not used. 1 Dedicated peripheral function. The signal 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.

33.3.2 Port B Configuration Example

PB[31] can be configured as a general-purpose I/O or open-drain signal. It can also be the $\overline{\text{REJECT1}}$ signal for the SCC1 Ethernet CAM interface or the SPI select input, $\overline{\text{SPISEL}}$. If PB[31] is not configured to connect to $\overline{\text{REJECT1}}$ or $\overline{\text{SPISEL}}$, the SCC and/or SPI receives V_{DD} on that signal. In the description of the PIP, PB[31] and other port B signals can be used as PIP functions. However, the PIP does not affect the operation of port B unless it is enabled.

33.4 Port C

Port C consists of 12 general-purpose I/O signals that can generate interrupts, which are managed by the CPM interrupt controller (CPIC). Table 33-11 lists port C signal options.

Table 33-11. Port C Pin Assignment

Signals	PCPAR[DD*n*] = 0		PCPAR[DD*n*] = 1		Input to On-Chip Peripherals (Default)
	PCDIR[DR*n*] = 1 or PCSO[*n*] = 0	PCDIR[DR*n*] = 0 and PCSO[*n*] = 1	PCDIR[DR*n*] = 0	PCDIR[DR*n*] = 1	
PC15	Port C15	$\overline{\text{DREQ0}}$	$\overline{\text{RTS1}}$	L1ST1	$\overline{\text{DREQ0}}$ = V_{DD}
PC14	Port C14	$\overline{\text{DREQ1}}$	—	L1ST2	$\overline{\text{DREQ1}}$ = V_{DD}
PC13	Port C13	—	$\overline{\text{RTS3}}$	L1ST3	—
PC12	Port C12		$\overline{\text{L1RQa}}$	L1ST4	—
PC11	Port C11	$\overline{\text{CTS1}}$	—		GND
PC10	Port C10	$\overline{\text{CD1}}$	$\overline{\text{TGATE1}}$		GND
PC9	Port C9	—	—		GND
PC8	Port C8	—	$\overline{\text{TGATE2}}$		GND
PC7	Port C7			$\overline{\text{SDACK2}}$	GND
PC6	Port C6		—		GND
PC5	Port C5	—	L1TSYNCA	$\overline{\text{SDACK1}}$	GND
PC4	Port C4	—	L1RSYNCA		= GND

PCDIR and PCPAR bits are cleared at system reset, making all port signals general-purpose inputs. The CPM interrupt mask register (CIMR) (see Section 34.5.3, “CPM Interrupt Mask Register”) is also cleared, so port C I/O signals left floating do not cause false interrupts.

General-purpose port C I/O signals can be accessed through PCDAT where written data is stored in an output latch. If a port C signal is configured as an output, output latch data is gated onto the port signal. Reading PCDAT reads the value of the port signal itself. For port C input signals, data written to PCDAT is stored in the output latch but cannot reach the port signal. In this case, when the PCDAT register is read, the state of the port signal is read.

The following steps configure port C signals as general-purpose outputs. When the signal is configured as an output, port C interrupts are not generated.

1. Write the corresponding PCPAR bit with a 0.
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’.
5. Write the signal value using the PCDAT register.

The following steps can be taken to configure a port C signal as a general-purpose input signal that does not generate an interrupt:

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 signal value using the PCDAT register.

When a port C signal 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 CPIC. Each port C signal can be configured to assert an interrupt request either when a high-to-low change occurs or when any change occurs. Each port C signal asserts a unique interrupt request to the CPM interrupt pending register (CIPM) (see Section 34.5.2, “CPM Interrupt Pending Register (CIPR)”) and has a different internal interrupt priority level within the CPM interrupt controller (see Section 34.2, “CPM Interrupt Source Priorities.”

Requests can be masked independently in the CPM interrupt mask register (CPMR). See Section 34.5.3, “CPM Interrupt Mask Register.” The following steps configure a port C signal as a general-purpose input that generates an interrupt:

1. Write the corresponding PCPAR bit with a 0.
2. Write the corresponding PCDIR bit with a 0.
3. Write the corresponding PCSO bit with a 0.
4. Set the PCINT bit to determine which edges cause interrupts.
5. Write the corresponding CIMR bit with a 1 so that interrupts can be sent to the core.
6. Read the signal value using the PCDAT register.

The port C signals associated with $\overline{\text{CDx}}$ and $\overline{\text{CTSx}}$ have a mode of operation in which the signal can be connected to the SCC internally but can also generate interrupts. Port C still detects changes on $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ and asserts the corresponding interrupt request, but the SCC simultaneously uses $\overline{\text{CTS}}$ and/or $\overline{\text{CD}}$ to control operation automatically. This allows the implementation of V.24, X.21, and X.21 bis protocols with help from other general-purpose I/O signals. To configure a port C signal as a $\overline{\text{CTS}}$ or $\overline{\text{CD}}$ signal that connects to the SCC and generates interrupts, follow these steps:

1. Write the corresponding PCPAR bit with a 0.
2. Write the corresponding PCDIR bit with a 0.
3. Write the corresponding PCSO bit with a 1.
4. Set the PCINT bit to determine which edges cause interrupts.
5. Write the corresponding CIMR bit with a 1 so that interrupts can be sent to the core.
6. The signal value can be read at any time using the PCDAT register.

After connecting $\overline{\text{CTS}}$ or $\overline{\text{CD}}$ to the SCC, choose normal operation mode in GSMR[DIAG] to enable or disable SCC transmission and reception with these signals.

PC14 and PC15 can be programmed to assert special requests directly to the CPM by setting RCCR[EIE]; however, do not do so unless instructed by a Motorola-supplied RAM microcode package.

For IDMA, PC14 and PC15 can be programmed to function as external DMA request ($\overline{\text{DREQ}}x$) signals. Do not configure PC14and PC15 as $\overline{\text{DREQ1}}$ and $\overline{\text{DREQ0}}$ unless IDMA is initialized; otherwise, erratic operation can occur.

33.4.1 Port C—RxClav Signal

Port C also includes the RxClav signal. When PDPAR[UT] is set, DREQ0[PC15] is configured to support the RxClav signal. The PCPAR and PCDIR fields must be cleared and the PCSO field must be set to enable the RxClav signal input on the PC15 signal.

33.4.2 Port C Registers

Port C is supported by five registers. The port C interrupt control register (PCINT) defines how changes on the signal cause interrupts when they are generated with that signal. The port C special options register (PCSO) determines whether certain port C signals can connect to on-chip peripherals and generate an interrupt at the same time. The remaining port C registers (PCDAT, PCDIR, and PCPAR) have the same functions as their counterparts on ports A and B. Port C has no open-drain capability.

33.4.2.1 Port C Data Register (PCDAT)

When read, the port C data (PCDAT) register always reflects the current status of each line.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—				D4–D15											
Reset	0															
R/W	R/W															
Addr	0x966															

Figure 33-11. Port C Data Register (PCDAT)

Table 33-12 describes PCDAT bits.

Table 33-12. PCDAT Bit Descriptions

Bits	Name	Description
0–3	—	Reserved
4–15	D*n*	Contains the data on the corresponding signal.

33.4.2.2 Port C Data Direction Register (PCDIR)

Port C data direction register (PCDIR) bits configure port C signals as general-purpose inputs or outputs. If a signal is not programmed for general-purpose I/O, PCDIR, along with PCSO, selects the peripheral function to be performed.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—				DR4–DR15											
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	0x960															

Figure 33-12. Port C Data Direction Register (PCDIR)

Table 33-13 describes PCDIR bits.

Table 33-13. PCDIR Bit Descriptions

Bits	Name	Description
0–3	—	Reserved
4–15	DR*n*	Port C data direction. Configures port C signals as inputs or outputs when functioning as general-purpose I/O; otherwise, used with PCSO to select the peripheral function. 0 Select the signal for general-purpose input, or select peripheral function 0. 1 Select the signal for general-purpose output, or select peripheral function 1.

33.4.2.3 Port C Pin Assignment Register (PCPAR)

The port C pin assignment register (PCPAR) configures signals as general-purpose I/O or dedicated for use with a peripheral.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—				DD4	DD5	DD6	DD7	DD8	DD9	DD10	DD11	DD12	DD13	DD14	DD15
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	0x962															

Figure 33-13. Port C Pin Assignment Register (PCPAR)

Table 33-14 describes PCPAR bits.

Table 33-14. PCPAR Bit Descriptions

Bits	Name	Description
0–3	—	Reserved.
4–15	DD*n*	Configures a signal for general-purpose I/O or for dedicated peripheral function 0 General-purpose I/O. The peripheral functions of the signal are not used. 1 Dedicated peripheral function. The signal is used by the internal module. The on-chip peripheral function to which it is dedicated can be determined by other bits.

33.4.2.4 Port C Special Options Register (PCSO)

The port C special options (PCSO) register, shown in Figure 33-14, further configures the corresponding port C signals.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—	—	—	—	—	—					CD1	CTS1	—	—	DREQ1	DREQ0
Reset	0															
R/W	R/W															
Addr	0x964															

Figure 33-14. Port C Special Options Register (PCSO)

Table 33-15 describes PCSO bits.

Table 33-15. PCSO Bit Descriptions

Bits	Name	Description
0–3	—	Reserved, should be cleared.
4, 810	CD*x*	Carrier detect 0 PC*x* is a general-purpose interrupt I/O signal. The SCC internal $\overline{\text{CD}}$*x* signal is always asserted. If PCDIR configures $\overline{\text{CD}}$*x* as an input, it can generate an interrupt to the core, as controlled by the PCINT bits. 1 PC*x* is connected to the corresponding SCC input as well as being a general-purpose interrupt signal.
5, 911	CTS*x*	Clear to send 0 PC*x* is a general-purpose interrupt I/O signal. The SCC internal $\overline{\text{CTS}}$*x* signal is always asserted. If PCDIR configures this signal as an input, the signal can generate an interrupt to the core, as controlled by the PCINT bits. 1 PC*x* is connected to the corresponding SCC input as well as being a general-purpose interrupt signal.
14–15	DREQ*x*	Enable DMA request to the CPM. Set DREQ*x* only if IDMA is being used. Note that the IDMA request function and the general-purpose interrupt function operate concurrently and independently. 0 PC*x* is a general-purpose interrupt I/O signal. If PCDIR configures this signal as an input, the signal can generate an interrupt to the core, as controlled by the PCINT bits. 1 As well as being a general-purpose interrupt signal PC*x* becomes an external request to the CPM for IDMA service. RCCR[DR*x*M] controls whether IDMA requests are edge- or level-sensitive. The corresponding PCINT bits still control when a general-purpose interrupt is generated.

33.4.2.5 Port C Interrupt Control Register (PCINT)

Each bit of the port C interrupt control (PCINT) register, shown in Figure 33-15, corresponds to a port C signal to determine whether that line asserts an interrupt request on

a high-to-low transition or on any transition. PCINT is cleared by reset.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—				EDM4–EDM15											
Reset	0															
R/W	R/W															
Addr	0x968															

Figure 33-15. Port C Interrupt Control Register (PCINT)

Table 33-16 describes PCINT bits.

Table 33-16. PCINT Bit Descriptions

Bits	Name	Description
0–3	—	Reserved and should be cleared.
4–15	EDM*n*	Edge detect mode. The corresponding port C signal asserts an interrupt request. 0 Any edge on PC*x* generates an interrupt request. 1 A falling edge on PC*x* generates an interrupt request.

33.5 Port D

The 13 port D signals are configured independently as general-purpose I/O signals if the corresponding port D pin assignment register (PDPAR) is cleared. They are configured as dedicated on-chip peripheral signals if the corresponding PDPAR bit is set.

The port I/O signal is configured as an input if the corresponding bit in the port D data direction register (PDDIR) is cleared and as an output if the bit is set. PDPAR and PDDIR are cleared at system reset, which configures all port D signals as general-purpose inputs. Table 33-17 describes port D signal defaults.

Table 33-17. Port D Pin Assignment

Signal	PDPAR = 0	PDPAR=1			Input to On-Chip Peripherals
		UT=0		UT=1	
		PDDIR=0	PDDIR=1		
PD15	Port D15	L1TSYNCA	MII-RXD3 (I)	UTPB[0]	L1TSYNCA=GND
PD14	Port D14	L1RSYNCA	MII-RXD2 (I)	UTPB[1]	L1RSYNCA=GND
PD13	Port D13	—	MII-RXD1 (I)	UTPB[2]	—
PD12	Port D12	—	MII-MDC (O)	UTPB[3]	—
PD11	Port D11	—	MII-TX-ERR (O)	RxEnb	—
PD10	Port D10	—	MII-RXD0 (I)	TxEnb	—
PD9	Port D9	—	MII-TXD0 (O)	UtpClk	—
PD8	Port D8	—	MII-RX_CLK (I)	—	—
PD7	Port D7	—	MII-RX-ERR(I)	UTPB[4]	—

Table 33-17. Port D Pin Assignment (continued)

PD6	Port D6	—	MII-RXDV (I)	UTPB[5]	—
PD5	Port D5	—	MII-TXD3 (O)	UTPB[6]	—
PD4	Port D4	—	MII-TXD2 (O)	UTPB[7]	—
PD3	Port D3	—	MII-TXD1 (O)	SOC	—

33.5.1 Port D Registers

Port D has three memory-mapped, read/write control registers.

33.5.1.1 Port D Data Register

A read of the port D data (PDDAT) register returns the value of the signal, regardless of whether it is an input or output. This allows output conflicts to be found on the signal by comparing the written data with the data on the signal. A write to a PDDAT bit is latched, and if configured as an output, is driven onto its respective signal. PDDAT can be read or written at any time. PDDAT is not initialized and is undefined by reset.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—			D3–D15												
Reset	Undefined															
R/W	R/W															
Addr	0x976															

Figure 33-16. Port D Data Register (PDDAT)

Table 33-18 describes PDDAT bits.

Table 33-18. PDDAT Bit Descriptions

Bits	Name	Description
0–2	—	Reserved
3–15	D*n*	Contains the data on the corresponding signal.

33.5.1.2 Port D Data Direction Register (PDDIR)

The port D data direction register (PDDIR) provides bits for specifying whether port D signals are inputs or outputs when functioning as general-purpose I/O.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—			DR3	DR4	DR5	DR6	DR7	DR8	DR9	DR10	DR11	DR12	DR13	DR14	DR15
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	0x970															

Figure 33-17. Port D Data Direction Register (PDDIR)

Table 33-19 describes PDDIR bits.

Table 33-19. PDDIR Bit Descriptions

Bits	Name	Description
0–2	—	Reserved and should be cleared.
3–15	DR*n*	Port D data direction. Configures port D signals as inputs or outputs when functioning as general-purpose I/O. 0 The corresponding signal is an input. 1 The corresponding signal is an output.

33.5.2 Port D Pin Assignment Register (PDPAR)

The ATM and UT bits are included in the PDPAR register, shown in Figure 33-18. The PDPAR register is cleared at system reset.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	ATM	UT	—	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
Oper	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	OFFSET TO IMMR: 0X972 (PDPAR)															

Figure 33-18. Port D Pin Assignment Register (PDPAR)

The fields in the PDPAR register are described in Table 33-20.

Table 33-20. PDPAR Field Descriptions

Bits	Name	Description
0	ATM	ATM global enable. 0 =Disable ATM SAR functionality 1 =Enable ATM SAR functionality
1	UT	UTOPIA enable. Determines whether the parameter RAM's page 4 operates in serial or UTOPIA mode. 0 =Serial mode using page 4. 1 = UTOPIA mode
2	—	Reserved
3–15	DDx	Signal assignment. Determines whether the signal is a general-purpose I/O signal or performs a dedicated function. 0 =General-purpose I/O. The peripheral functions of the signal are not used. 1 =Dedicated peripheral function. The signal performs the function assigned by the internal module.

Chapter 34
CPM Interrupt Controller

The CPM interrupt controller (CPIC) accepts and prioritizes the internal and external interrupt requests from the CPM blocks and passes them to the system interface unit (SIU). The CPIC also provides a vector during the core interrupt acknowledge cycle.

34.1 Features

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

- Twenty-six interrupt sources—14 internal and 12 external (through port C)
- Sources can be assigned to a programmable interrupt level
- Programmable highest priority request
- Fully-nested interrupt environment
- Individual interrupt sources can be masked in the CPM interrupt mask register (CIMR).
- Unique vector number for each interrupt source

The CPIC manages interrupts from internal CPM sources. These sources are primarily generated by controllers, such as SCC1, SMCs, SPI, and I^2C but also include the 12 general-purpose timers and port C parallel I/O signals described in Section 33.4, "Port C." More than one of these sources may generate interrupts at the same time; therefore, the CIMR register is provided for masking individual sources. Additional masking is provided for specific interrupt events within each controller that reports interrupts through the CPIC. These mask registers are described in the chapters that describe individual controllers. All CPIC-managed interrupt sources are prioritized and bits are set in the CPM interrupt pending register (CIPR).

Figure 34-1 shows the MPC855T interrupt structure. The left of the figure shows individual interrupt sources managed by the CPIC, which signals CPIC-managed interrupts to the SIU, shown in the middle of Figure 34-1. All interrupts signaled by the CPIC are presented to the SIU at a single programmable priority level (0–7). In turn, the SIU controls which PowerPC architecture-defined external interrupt exception condition is reported to the MPC8xx core.

For information about the SIU interrupt structure, see Section 10.5.1, "Interrupt Structure." For information about the external interrupt exception, see Section 6.1.2.5, "External Interrupt Exception (0x00500)."

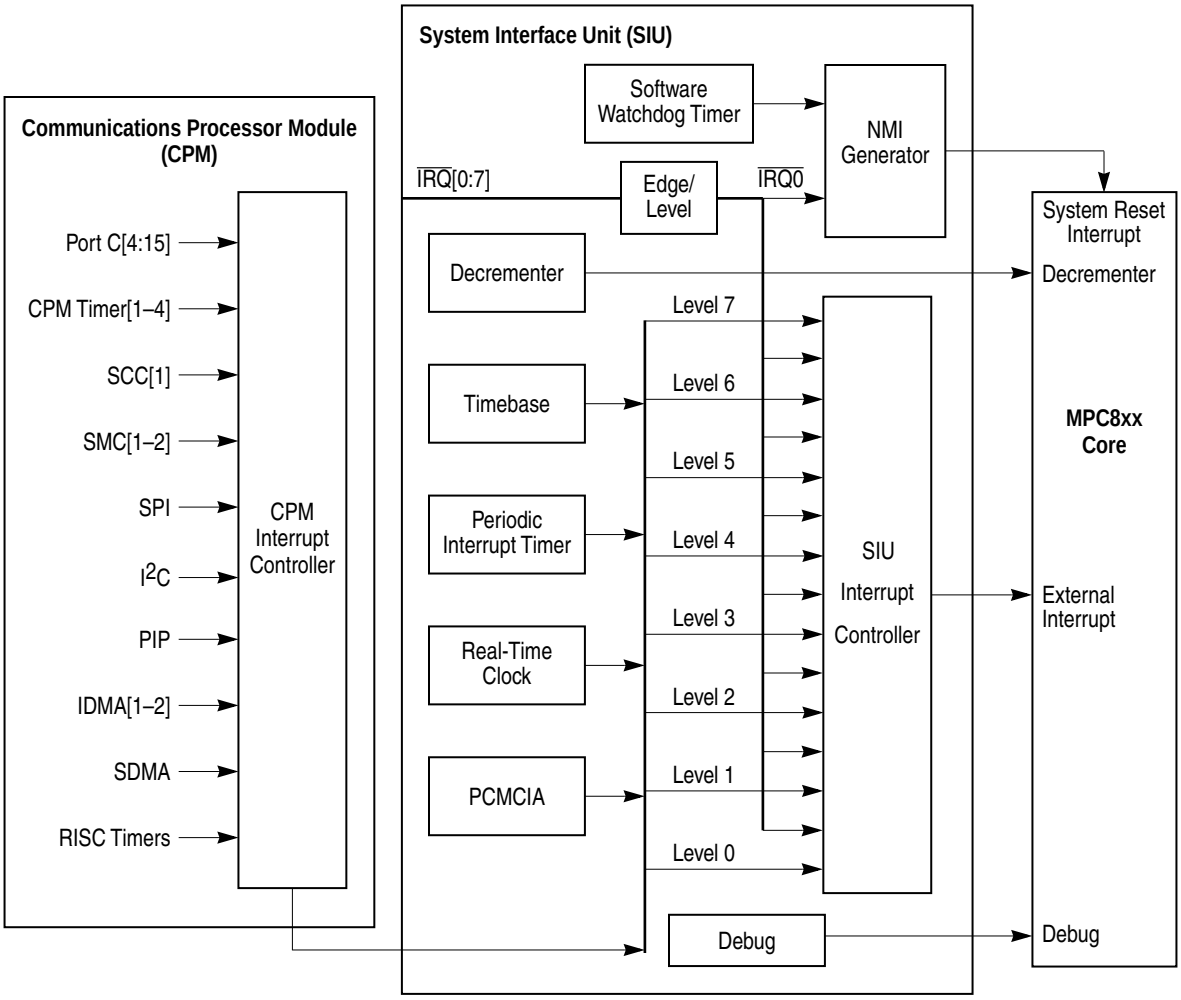


Figure 34-1. MPC855T Interrupt Structure

Although all CPM interrupts are presented to the SIU at the same priority level (specified in CICR[IRL]), individual CPM interrupt sources are prioritized as described in Section 34.2, “CPM Interrupt Source Priorities.” The MPC855Tprovides limited ability to reorder the interrupt priorities of SCCs and to specify the highest priority interrupt source.

As shown in Figure 34-1, when the CIPR indicates that an unmasked interrupt source is pending, the CPIC sends an interrupt request to the SIU at the interrupt level specified in CICR[IRL]. The CPIC then waits for the interrupt to be recognized. The core honors the interrupt request and then acknowledges the interrupt by setting the IACK bit in the CPM interrupt vector register (CIVR). When CIVR[IACK] is set, the contents of CIVR[VN] are updated with the 5-bit vector corresponding to the sub-block with the highest current priority. CIVR[IACK] is cleared after one clock cycle.

34.2 CPM Interrupt Source Priorities

The CPIC has interrupt sources that assert a single programmable interrupt request level to the core. Default interrupt priorities are as shown in Table 34-1.

Table 34-1. Prioritization of CPM Interrupt Sources

Priority	Source Description	Multiple Events	Priority	Source Description	Multiple Events
0x1F (Highest)	Parallel I/O–PC15 [1]	No	0x0F	Parallel I/O–PC11 [1]	No
0x1E	SCC	Yes	0x0E	Parallel I/O–PC10 [1]	No
0x1D	SCC	Yes	0x0D	SCC	Yes
0x1C	SCC	Yes	0x0C	Timer3	Yes
0x1B	SCC	Yes	0x0B	Parallel I/O–PC9 [1]	No
0x1A	Parallel I/O–PC14 [1]	No	0x0A	Parallel I/O–PC8 [1]	No
0x19	Timer 1	Yes	0x09	Parallel I/O–PC7 [1]	No
0x18	Parallel I/O–PC13 [1]	No	0x08	SCC	Yes
0x17	Parallel I/O–PC12 [1]	No	0x07	Timer4	Yes
0x16	SDMA channel bus error	Yes	0x06	Parallel I/O–PC6 [1]	No
0x15	IDMA1	Yes	0x05	SPI	Yes
0x14	IDMA2	Yes	0x04	SMC1	Yes
0x13	SCC	Yes	0x03	SMC2	Yes
0x12	Timer 2	Yes	0x02	Parallel I/O–PC5 [1]	No
0x11	RISC timer table	Yes	0x01	Parallel I/O–PC4 [1]	No
0x10	I^2C	Yes	0x00 (Lowest)	Reserved	—

[1] Port C interrupts (external sources) are described in Section 33.4.2.5, “Port C Interrupt Control Register (PCINT).”

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. Other SDMA-related interrupts are reported through each individual SCC, SMC, SPI, or I^2C channel.

34.2.1 Highest Priority Interrupt

The highest priority interrupt source can be selected dynamically by entering the interrupt number in CICR[HP*n*], described in Table 34-3. This interrupt is still within the same interrupt level specified in CICR[IRL] but is serviced before any other CPM interrupt (that is, if this type of interrupt is pending, its vector number returns first when the CIVR is read.

34.2.2 Nested Interrupts

The CPIC supports a fully nested interrupt environment that allows a high priority interrupt from another CPM source to suspend a lower priority service routine. An interrupt request with highest priority is presented to the core for servicing, which the core acknowledges by setting CIVR[IACK]. After IACK is set, the corresponding vector is indicated in the CIVR and the request is cleared. The next request can be presented to the core. When the interrupt is taken, the external interrupt enable bit of the core's machine state register, MSR[EE] is cleared to disable further interrupt requests until software can handle them.

The CPM interrupt in-service register (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 completes. Each CISR bit corresponds to a CPM interrupt source. When the core acknowledges the interrupt by setting IACK, the CPIC sets the CISR bit for that interrupt source. This prevents subsequent CPM interrupt requests at this priority level or lower, until the current interrupt is serviced and the CISR bit is cleared. Lower-priority interrupts can still be set in the CPIC during this time, but they will pend until the CISR bit for the higher-priority interrupt is cleared. Therefore, in the interrupt service routine for the CPM interrupts, the core external interrupt enable MSR[EE] can be set to allow higher-priority interrupts within the CPM or from other sources to generate an interrupt request.

34.3 Masking Interrupt Sources in the CPM

An interrupt is masked by clearing and enabled by setting the corresponding CIMR bit; see Section 34.5.3, "CPM Interrupt Mask Register." When a masked source requests an interrupt, the corresponding CIPR bit is set but the CPIC does not signal the interrupt to the core. Masking all sources allows the implementation of a polling interrupt servicing scheme.

CPM sub-blocks with multiple interrupting events can be masked individually by programming a mask register within that block (such as the SMC UART register (SMCM), described in Section 29.3.12, "SMC UART Event Register (SMCE)/Mask Register (SMCM)"). Table 34-2 shows the interrupt sources that have multiple interrupting events. Figure 34-2 shows masking using the SMC sub-block.

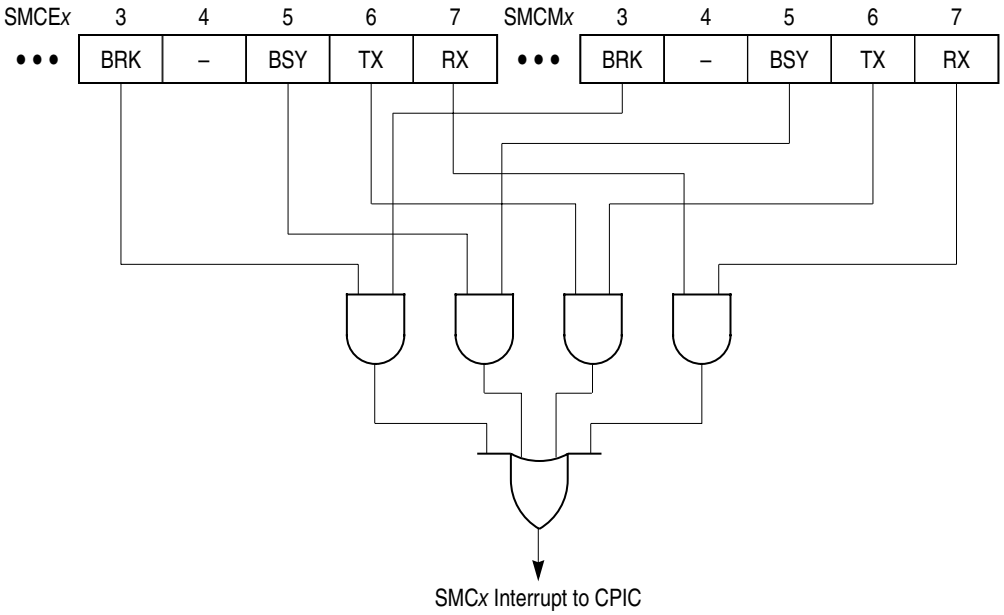


Figure 34-2. Interrupt Request Masking

The following procedure prevents possible interrupt errors when modifying mask registers, such as the CIMR, SCCM, SMCM, or any other CPM interrupt mask:

1. Clear MSR[EE]. (Disable external interrupts to the core.)
2. Modify the mask register.
3. Set MSR[EE]. (Enable external interrupts to the core.)

This mask modification procedure ensures that an already pending interrupt is not masked before being serviced. Masking a pending interrupt causes the interrupt error vector (see Table 34-2) to be issued if no other valid CPM interrupts are pending. (The error vector cannot be masked.)

34.4 Generating and Calculating Interrupt Vectors

Unmasked CPM interrupts are presented to the core in order of priority. The core responds to an interrupt request by setting CIVR[IACK]. The CPIC passes the five low-order bits of the vector corresponding to the highest priority, unmasked, pending CPM interrupt in CIVR[VN]. These encodings are shown in Table 34-2.

Table 34-2. Interrupt Vector Encodings

Interrupt Number	Source Description	CIVR[0–4]	Interrupt Number	Source Description	CIVR[0–4]
0x1F	Parallel I/O–PC15	11111	0x0F	Parallel I/O—PC11	01111
0x1E	SCC1	11110	0x0E	Parallel I/O—PC10	01110

Table 34-2. Interrupt Vector Encodings (continued)

Interrupt Number	Source Description	CIVR[0–4]	Interrupt Number	Source Description	CIVR[0–4]
0x1D	—	11101	0x0D	Reserved	01101
0x1C	—	11100	0x0C	Timer 3	01100
0x1B	—	11011	0x0B	Parallel I/O—PC9	01011
0x1A	Parallel I/O—PC14	11010	0x0A	Parallel I/O—PC8	01010
0x19	Timer 1	11001	0x09	Parallel I/O—PC7	01001
0x18	Parallel I/O—PC13	11000	0x08	Reserved	01000
0x17	Parallel I/O—PC12	10111	0x07	Timer 4	00111
0x16	SDMA channel bus error	10110	0x06	Parallel I/O—PC6	00110
0x15	IDMA1	10101	0x05	SPI	00101
0x14	IDMA2	10100	0x04	SMC1	00100
0x13	Reserved	10011	0x03	SMC2	00011
0x12	Timer 2	10010	0x02	Parallel I/O—PC5	00010
0x11	RISC timer table	10001	0x01	Parallel I/O—PC4	00001
0x10	I^2C	10000	0x00	Error	00000

The table is the same as the CPM interrupt priority table (Table 34-1) except that the SCC vector number is fixed. Also the last entry in this table is the error vector, which the CPM issues if it requested an interrupt that the user cleared before the core serviced it and no other interrupts for the CPM are pending. The user should provide an error interrupt service routine even if it is only an **rfi** instruction.

34.5 CPIC Registers

There are four CPIC registers:

- CPM interrupt configuration register (CICR)—Defines CPM interrupt attributes.
- CPM interrupt pending register (CIPR)—Indicates which CPM interrupt sources require interrupt service.
- CPM interrupt mask register (CIMR)—Can be used to mask CPM interrupt sources.
- CPM interrupt in-service register (CISR)—Allows nesting interrupt requests within the CPM interrupt level.

Note that the names and placement of bits is identical in the CIPR, CIMR, and CISR.

34.5.1 CPM Interrupt Configuration Register (CICR)

The CPM interrupt configuration register (CICR) defines CPM interrupt request levels, the priority between the SCCs, and the highest priority interrupt.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Reset	0000_0000_0000_0000															
R/W	R/W															
Address	0x940															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	IRL			HP					IEN	—						—
Reset	0000_0000_0000_0000															
R/W	R/W															
Add	0x942															

Figure 34-3. CPM Interrupt Configuration Register (CICR)

This register is affected by $\overline{\text{HRESET}}$ but is not affected by $\overline{\text{SRESET}}$. CICR bits are described in Table 34-3.

Table 34-3. CICR Field Descriptions

Bits	Name	Description
0–15	—	Reserved, should be cleared.
16–18	IRL	Interrupt request level. Contains the priority request level of the interrupt from the CPM that is sent to the SIU. Level 0 indicates highest priority. IRL is initialized to zero during reset. In most systems, value 0b100 is a good value to choose for IRL.
19–23	HP	Highest priority. Specifies the 5-bit interrupt number of the CPIC interrupt source that is advanced to the highest priority in the table. These bits can be modified dynamically. (Programming HP = 0b11111 keeps PC15 the highest priority source for external interrupts to the core.)
24	IEN	Interrupt enable. Master enable for CPM interrupts. 0 CPM interrupts are disabled 1 CPM interrupts are enabled
25–30	—	Reserved
31	—	Reserved

34.5.2 CPM Interrupt Pending Register (CIPR)

Each bit in the read/write CPM interrupt pending register (CIPR) corresponds to a CPM interrupt source. The CPIC sets the appropriate CIPR bit when a CPM interrupt is received. Names and placement of bits, shown in Figure 34-4, are identical in the CIPR, CIMR, and CISR, and they follow the priorities described in Table 34-1. These registers are affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	PC15	SCC1	—	—	—	PC14	TIMER1	PC13	PC12	SDMA	IDMA1	IDMA2	—	TIMER2	RTT	I2C
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	0x944 (CIPR), 0x948 (CIMR), 0x94C (CISR)															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	PC11	PC10	—	TIMER3	PC9	PC8	PC7	—	TIMER4	PC6	SPI	SMC1	SMC2/ PIP	PC5	PC4	—
Reset	0000_0000_0000_0000															
R/W	R/W															
Addr	0x946 (CIPR), 0x94A (CIMR), 0x94E (CISR)															

Figure 34-4. CPM Interrupt Pending/Mask/In-Service Registers (CIPR/CIMR/CISR)

In a vectored interrupt scheme, the CPIC clears the appropriate CIPR bit when the core acknowledges the interrupt by setting CIVR[IACK]. The vector number corresponding to the CPM interrupt source is then available for the core in CIVR[VN]. However, the CIPR bit is not cleared if an event register exists for that interrupt source. Event registers exist only for interrupt sources with multiple interrupt events.

In a polled interrupt scheme, the user must periodically read the CIPR. To avoid losing subsequent events from the same interrupt source, acknowledge an interrupt before actually handling it in the service routine. Acknowledge interrupts from port C by clearing the CIPR bit directly (by writing ones). For all other interrupt sources, however, clear the unmasked event register bits instead, thus causing the CIPR bit to be cleared.

34.5.3 CPM Interrupt Mask Register

Each bit in the read/write CPM interrupt mask register (CIMR) corresponds to a CPM interrupt source indicated in CIPR. The CIPR and CIMR are shown in Figure 34-4. An interrupt is masked by clearing and enabled by setting the corresponding CIMR bit. Even if an interrupt is masked, the corresponding CIPR bit is set when an interrupt condition occurs, but the interrupt request is not passed to the core.

If a CPM interrupt source is requesting interrupt service when its CIMR bit is cleared, the request stops. If the bit is set later, the core processes previously pending interrupt requests according to priority.

34.5.4 CPM Interrupt In-Service Register (CISR)

Each bit in the CPM interrupt in-service register (CISR) corresponds to a CPM interrupt source. The CISR, CIPR, and CIMR are shown in Figure 34-4. In a vectored interrupt environment, the CPIC sets a CISR bit when the core acknowledges the interrupt by setting

CIVR[IACK]. An interrupt service routine must clear the corresponding CISR bit after servicing is complete. If an event register exists for this peripheral, its bits would normally be cleared. Write ones to clear CISR bits; writing zeros has no effect.

Bits set in this register indicate which interrupt requests are in progress for each CPM interrupt source. More than one CISR bit can be set 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 TIMER2 interrupt handler. See Section 34.2.2, “Nested Interrupts.” During this time, both CISR[TIMER1] and CISR[TIMER2] are set.

If error vector is taken, no CISR bit is set. All undefined CISR bits return zeros when read. The extent to which CPM interrupts can interrupt one another is controlled by selectively clearing the CISR. A new interrupt is processed if it has a higher priority than the highest priority interrupt having its CISR bit set. Thus, if an interrupt routine sets the external interrupt enable bit in the core (MSR[EE]) and clears its CISR bit at the beginning of the interrupt routine, a lower priority interrupt can interrupt a higher one if the lower priority interrupt has higher priority than any other CISR bits that are currently set. Therefore, the interrupt service routine should clear its CISR bit at the end.

34.5.5 CPM Interrupt Vector Register (CIVR)

The CPM interrupt vector register (CIVR) is used to identify an interrupt source. The core uses the IACK bit to acknowledge an interrupt. CIVR can be read at any time. This register is affected by $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	VN					0										IACK
Reset	0000_00000_0000_0000															
R/w	R/W															
Address	0x930															

Figure 34-5. CPM Interrupt Vector Register (CIVR)

Table 34-4 describes CIVR fields. Section 34.6, “Interrupt Handler Example—Single-Event Interrupt Source,” and Section 34.7, “Interrupt Handler Example—Multiple-Event Interrupt Source,” show how CIVR fields are used.

Table 34-4. CIVR Field Descriptions

Bits	Name	Description
0–4	VN	Vector number. Identifies the interrupt source. These values are listed in Table 34-2.
5–14	—	Reserved. Writing to bits 5-15 has no effect because they are always read as zeros.
15	IACK	Interrupt acknowledge. When the core sets IACK, CIVR[VN] is updated with a 5-bit vector corresponding to the sub-block with the highest current priority. IACK is cleared after one clock cycle.

34.6 Interrupt Handler Example—Single-Event Interrupt Source

In this example, the CPIC hardware clears CIPR[PC6] during the interrupt acknowledge cycle. The following steps show how to handle an interrupt source without multiple events.

1. Set CIVR[IACK].
2. Read CIVR[VN] to determine the vector number for the interrupt handler.
3. Handle the interrupt event indicated through the port C6 signal.
4. Clear CISR[PC6].
5. Execute the **rfi** instruction.

34.7 Interrupt Handler Example—Multiple-Event Interrupt Source

In this example, CIPR[SCC1] remains set as long as one or more event bits remain unmasked in SCCE1. This is an example of a handler for an interrupt source with multiple events. Notice that the handler must clear the CISR bit but not the CIPR bit.

1. Set the CIVR[IACK].
2. Read CIVR[VN] to determine the vector number for the interrupt handler.
3. Immediately read the SCC1 event register into a temporary location.
4. Decide which events in the SCCE1 must be handled and clear those bits as soon as possible. SCCE bits are cleared by writing ones.
5. Handle the events in the SCC1 Rx BD or Tx BD tables.
6. Clear CISR[SCC2].
7. Execute the **rfi** instruction. If any unmasked SCCE bits remain (either not cleared by the software or set by the MPC855T during the execution of this handler), this interrupt source is pending again immediately after the **rfi** instruction.

Part VI
Asynchronous Transfer Mode (ATM)

Intended Audience

Part VI is intended for system designers who need to use the MPC8asynchronous transfer mode capabilities. It assumes a basic understanding of the PowerPC exception model, the MPC8interrupt structure, the MPC8communications processor module (CPM) with a particular emphasis on the SCC, as well a working knowledge of ATM. A complete discussion of these protocols is beyond the scope of this book.

Contents

Part VI describes the MPC8's implementation of asynchronous transfer mode (ATM). It contains the following chapters:

- Chapter 35, "ATM Overview," gives a high-level description of the MPC855TATM implementation
- Chapter 36, "Buffer Descriptors and Connection Tables," describes the structure and configuration of the buffer descriptors (BDs) and the transmit and receive connection tables (TCTs and RCTs) used with ATM.
- Chapter 37, "ATM Parameter RAM," describes how the parameter RAM is used to configure the SCC for serial ATM and the UTOPIA interface. The CP also uses parameter RAM to store operational and temporary values used during SAR activities.
- Chapter 38, "ATM Controller," describes the address mapping mechanisms of the ATM controller to support connection tables for single-PHY interfaces, and the commands provided to control ATM transmit and receive operations on a channel-by-channel basis.
- Chapter 39, "ATM Pace Control," describes how the ATM pace control unit (APC) processes traffic parameters of each channel and defines the multiplex timing for all the channels.

- Chapter 40, “ATM Exceptions,” describes how the circular ATM interrupt queue operates with an event register (SCCE or IDSR1) to provide an interrupt model for ATM operations.
- Chapter 41, “Interface Configuration,” describes the programming of registers and parameters for ATM operations through both the UTOPIA and serial interfaces.
- Chapter 42, “UTOPIA Interface,” describes how the MPC855Tsupports SAR MPHY ATM operation, including the UTOPIA modes and the signals provided for UTOPIA support.

Conventions

This document uses the following notational conventions:

Bold	Bold entries in figures and tables showing registers and parameter RAM should be initialized by the user.
mnemonics	Instruction mnemonics are shown in lowercase bold.
italics	Italics indicate variable command parameters, for example, **bcctr***x*. Book titles in text are set in italics.
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote binary number
rA, **r**B	Instruction syntax used to identify a source GPR
rD	Instruction syntax used to identify a destination GPR
REG[FIELD]	Abbreviations or acronyms for registers or buffer descriptors are shown in uppercase text. Specific bits, fields, or numerical ranges appear in brackets. For example, MSR[LE] refers to the little-endian mode enable bit in the machine state register.
x	In certain contexts, such as in a signal encoding or a bit field, indicates a don’t care.
n	Indicates an undefined numerical value
¬	NOT logical operator
&	AND logical operator
|	OR logical operator

Acronyms and Abbreviations

Table i contains acronyms and abbreviations used in this document. Note that the meanings for some acronyms (such as SDR1 and DSISR) are historical, and the words for which an acronym stands may not be intuitively obvious.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
AAL	ATM adaptation layer
AAL5	CPCS–PDU
ABR	Available bit rate
ACR	Allowed cell rate
ALU	Arithmetic logic unit
APC	ATM pace control
ATM	Asynchronous transfer mode
BD	Buffer descriptor
BIP	Bit interleaved parity
BIST	Built-in self test
BRC	Backward reporting cells
BT	Burst tolerance
CBR	Constant bit rate
CAM	Content-addressable memory
CEPT	Conference des administrations Europeanes des Postes et Telecommunications (European Conference of Postal and Telecommunications Administrations).
C/I	Condition/indication channel used in the GCI protocol
CLP	Cell loss priority
CP	Communications processor
CP-CS	Common part convergence sublayer
CPCS-PDU	Common part convergence sublayer–protocol data unit
CPCS-UU	Common part convergence sublayer–user to user information
CPI	Common part indicators
CPM	Communications processor module
CPS	Cells per slot
CSMA	Carrier sense multiple access
CSMA/CD	Carrier sense multiple access with collision detection
DMA	Direct memory access
DPLL	Digital phase-locked loop
DPR	Dual-port RAM
DRAM	Dynamic random access memory
DSISR	Register used for determining the source of a DSI exception

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
EA	Effective address
EEST	Enhanced Ethernet serial transceiver
EPROM	Erasable programmable read-only memory
ESAR	Enhanced segmentation and reassembly
FBP	Free buffer pool
FIFO	First-in-first-out (buffer)
FMC	Forward monitor cells
FRM	Forward resource management
GCI	General circuit interface
GCRA	Generic cell rate algorithm (leaky bucket)
GFC	Generic flow control
GPCM	General-purpose chip-select machine
GUI	Graphical user interface
HDLC	High-level data link control
HEC	Header error control
I^2C	Inter-integrated circuit
IDL	Inter-chip digital link
IEEE	Institute of Electrical and Electronics Engineers
IrDA	Infrared Data Association
ISDN	Integrated services digital network
JTAG	Joint Test Action Group
JTAG	Joint Test Action Group
LAN	Local area network
LIFO	Last-in-first-out
LRU	Least recently used
LSB	Least-significant byte
lsb	Least-significant bit
MAC	Multiply accumulate or media access control
MBS	Maximum burst size
MII	Media-independent interface
MSB	Most-significant byte

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
msb	Most-significant bit
MSR	Machine state register
NaN	Not a number
NCITS	Number of cells in a time slot
NIC	Network interface card
NIU	Network interface unit
NMSI	Nonmultiplexed serial interface
NRT	Non-real time
OSI	Open systems interconnection
PCI	Peripheral component interconnect
PDU	Protocol data unit
PCR	Peak cell rate
PHY	Physical layer
PM	Performance monitors
PPM	Pulse-position modulation
PTI	Payload type identifier
PTP	Port-to-port switching
RCT	Receive connection table
RM	Resource management
RT	Real-time
RTOS	Real-time operating system
Rx	Receive
SAR	Segmentation and reassembly
SCC	Serial communications controller
SCP	Serial control port
SCR	Sustained cell rate
SDLC	Synchronous Data Link Control
SDMA	Serial DMA
SI	Serial interface
SIU	System interface unit
SMC	Serial management controller

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
SNA	Systems network architecture
SPI	Serial peripheral interface
SRAM	Static random access memory
SRTS	Synchronous residual time stamp
TCT	Transmit connection table
TDM	Time-division multiplexed
TE	Terminal endpoint of an ISDN connection
TLB	Translation lookaside buffer
TSA	Time-slot assigner
Tx	Transmit
UBR	Unspecified bit rate
UBR+	Unspecified bit rate with minimum cell rate guarantee
UART	Universal asynchronous receiver/transmitter
UPM	User-programmable machine
USART	Universal synchronous/asynchronous receiver/transmitter
Utopia	Universal test operation physical interface for ATM
VBR	Variable bit rate
VC	Virtual channel or circuit or call or connection
VCC	Virtual channel connection
VCI	Virtual circuit identifier
VP	Virtual path
VPC	Virtual path connection
VPI	Virtual path identifier
UTOPIA	Universal test and operations physical interface for ATM
VC	Virtual channel or virtual circuit
WAN	Wide area network

Chapter 35
ATM Overview

This chapter provides an overview of the ATM features of the MPC855T.

35.1 ATM Capabilities

The MPC855T can be used as an adaptable ATM controller suited for a variety of applications, including the following:

- ATM line card controllers
- ATM to WAN interworking, including frame relay, T1/E1 circuit emulation service (CES), and xDSL applications
- Residential broadband network interface units (ATM-to-Ethernet)
- Set-top controllers
- ATM25 SAR applications
- Bridging and routing applications

35.2 MPC855T and MPC860 Differences

The MPC855T is pin-compatible with the standard MPC860, and they have identical electrical and mechanical specifications. However, when running an ATM application, the MPC855T loses some functionality due to internal resource conflicts.

35.2.1 Parameter RAM Conflicts

Operating serial ATM on SCC1 or the UTOPIA interface causes other peripherals to lose their parameter RAM. For SCC1, the serial ATM parameters extend into the I^2C parameter RAM default location. However, the parameters for both SPI and I^2C can be relocated without the need for RAM-based microcode. The parameter RAM for SMC2/PIP is affected in the same manner when using the UTOPIA interface.

35.2.2 IDMA2 Restriction

Because of internal system constraints, IDMA2 can only be used in level-sensitive mode when ATM is enabled. This requires the user to set the RCCR[DR1M] bit.

35.2.3 UTOPIA Conflicts

The UTOPIA interface is implemented using the hardware of IDMA1. Therefore, if the UTOPIA port is used:

- IDMA1 is unavailable. (The $\overline{\text{DREQ0}}$ signal is lost, and the IDMA1 event and mask registers IDSR1 and IDMR1 are used for UTOPIA events.)
- Parallel interface port (PIP) is unavailable (due to the loss of the PIP handshake signals)

35.2.4 The ATM Pace Controller (APC) and APC Timer

The ATM pace controller (APC) uses the CPM general-purpose timer 4 to schedule ATM cells for transmission. That is, CPM timer 4 becomes the APC timer and cannot be used as a general purpose timer. The APC can be disabled simply by disabling the APC timer.

35.3 ATM Features

The MPC855T supports the following features:

- Serial ATM capability on the SCC
- Optional UTOPIA port
- Cell processing up to 96 Mbps aggregate receive and transmit via UTOPIA interface (with 80 MHz system clock)
- Memory-to-memory cell processing (via UTOPIA or serial interface with loopback).
- Performs transmission convergence (TC) to E1/T1/xDSL serial lines.
- Support of AAL0 and AAL5 protocols on a per virtual circuit (VC) basis.
- AAL0 support allows other AAL types to be implemented in application software.
- Support for 32 active VCs using internal dual-port RAM, and up to 64K using external memory.
- Flexible and efficient cell rate pacing support for CBR and UBR, with software hooks provided for host-managed ABR services.
- Supports UTOPIA and serial (E1/T1/xDSL) interfaces.
- Compliant with ATM Forum UNI 4.0 specification.
- CLP and congestion indication marking in RxBD.

- Separate transmit and receive buffer descriptor (BD) tables for each channel.
- Interrupt reporting optionally enabled per channel
- Supports 53-byte to 64-byte (expanded) ATM cell size.
- Glueless serial interface to an xDSL interface device.
- Supports AAL5 connections:
  - Reassembly:
    - Reassembles CPCS_PDU directly to host memory
    - CRC32 check
    - CPCS_PDU padding removal
    - CS_UU, CPI, and LENGTH reporting
    - CLP and congestion reporting
    - Interrupt per buffer or per frame
    - Report on errors (CRC, length mismatch, message abort)
    - Real-time time stamp capability to support connection timeout
  - Segmentation:
    - Segments CPCS_PDU directly from host memory
    - Performs CPCS_PDU padding
    - CRC32 generation
    - Automatic last cell marking (in the PTI field of the cell header)
    - Automatic CS_UU, CPI, and LENGTH insertion (in the last cell of the frame)
- Supports AAL0 connections:
  - Receive:
    - Complete cell is stored in memory (with exception of the HEC).
    - CRC10 pass/fail indication
    - Optional interrupt per cell
  - Transmit:
    - Reads a complete cell (with exception of the HEC) from the buffer.
    - Optional CRC10 insertion
- Physical layer (PHY) support:
  - UTOPIA port provides glueless interface to PHY
    - UTOPIA master (ATM side) with cell-level handshake
    - Supports use of external logic to implement UTOPIA level 2 multi-PHY interface (for up to 4 PHY)
  - Serial interface
    - Cell delineation
    - HEC generation/checking
    - Cell payload scrambling/de-scrambling option ($X^{43}+1$ polynomial)

  - Automatic idle/un-assigned cell insertion/stripping
  - Cells with incorrect HEC are marked and counted.
- ATM pace control (APC) unit:
  - Constant bit rate (CBR) service on a per VC basis
  - Unspecified bit rate (UBR) pacing
  - Available bit rate (ABR) pacing (pace is managed by upper-layer host software when establishing a connection)
- Receive address mapping supported by three mechanisms:
  - Sequential look-up table (for up to 32 channels)
  - Flexible, user-defined address compression mechanism
  - Content-addressable memory (CAM)

35.4 MPC855T Application Example

Figure 35-1 shows a possible MPC855T configuration supporting both serial and UTOPIA ATM transmissions.

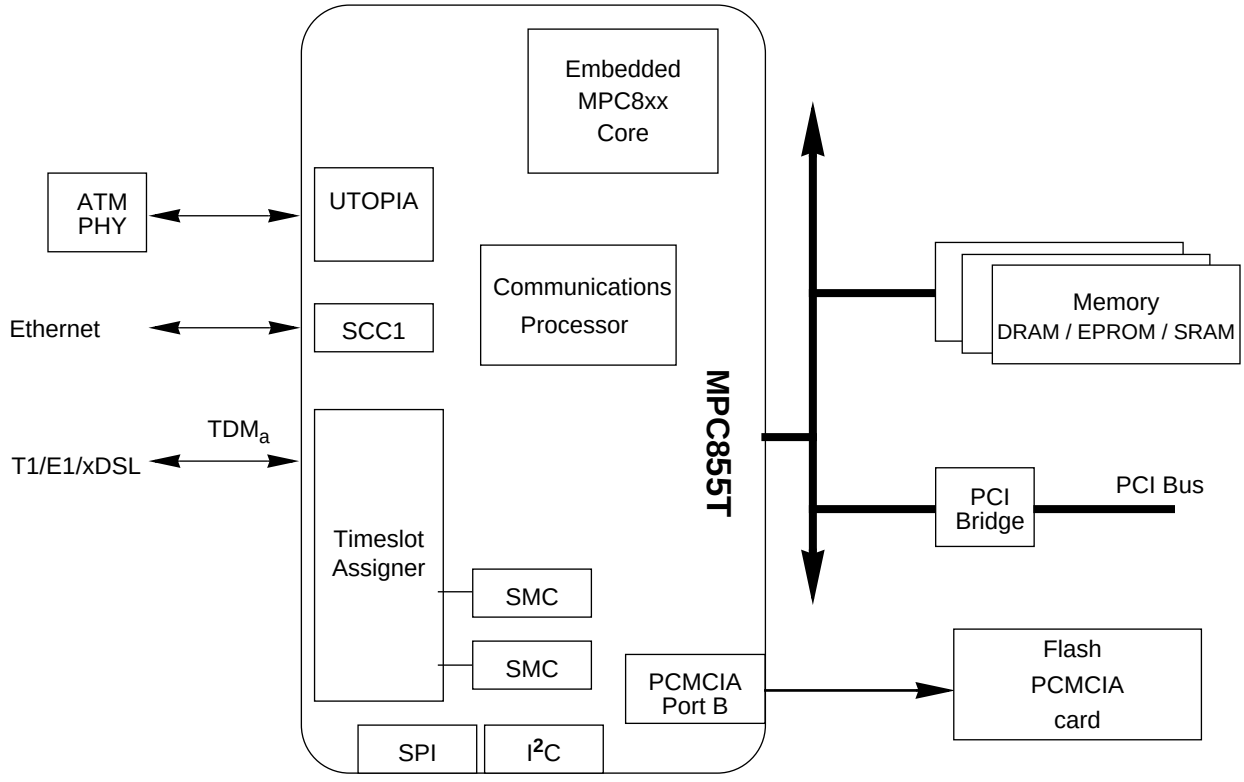


Figure 35-1. MPC855T Application Example

35.5 Overview of ATM Operation

The MPC855T supports ATM adaptation layers AAL0 and AAL5 segmentation and reassembly and the ATM layer for the convergence sublayer (CS). User data resides in system memory in single or multiple data buffers.

There are two physical layer/interface modes of operation: a UTOPIA interface and a serial interface. In UTOPIA mode, the ATM layer directly interfaces to the PHY through the UTOPIA interface. In serial mode, the ATM controller also implements the transmission convergence (TC) sublayer and interfaces to the PHY layer through the SCC.

The following sections describe the transfer mechanisms for the serial and UTOPIA interface modes and the functionality of the ATM pace controller (APC), which is utilized in both modes of operations. Internal and external ATM channels are introduced. Port-to-port cell switching and memory-to-memory SAR operation is also discussed.

35.6 UTOPIA Operation

In UTOPIA mode, the ATM controller handles transfers on a cell-by-cell basis. The UTOPIA interface implements a cell-level handshake. The supported bit rate of the UTOPIA interface is higher than that of serial ATM which additionally implements the transmission convergence (TC) layer.

The following sections describe the transmit and receive mechanisms for the UTOPIA interface. The expanded cell option is also discussed. Detailed information about UTOPIA mode can be found in Chapter 41, “Interface Configuration,” and in Chapter 42, “UTOPIA Interface.”

35.6.1 UTOPIA Transmit Overview

The UTOPIA transmit process begins with the ATM pace control unit (APC). The APC schedules the ATM traffic using a scheduling table and a user-configured APC timer (CPM general-purpose timer 4) that defines the maximum transmit bit rate (bandwidth). The APC maintains the traffic parameters for each channel and divides the total bandwidth among the active channels. It can provide CBR and UBR traffic services. ABR can also be supported through application software manipulation of APC parameters. See Chapter 39, “ATM Pace Control,” for additional information about the operation and programming of the APC.

With each tick of the APC timer, the APC prepares the channel(s) in the current time slot for transmission by inserting the channel number(s) into the transmit queue. When the PHY asserts the transmit cell available (TxClav) signal, the transmitter takes the next channel number from the transmit queue. The transmitter uses the channel number to find the channel’s transmit connection table (TCT).

For AAL5, the transmitter then copies 48 bytes (or up to 65 bytes for channels configured with expanded cells) from the external buffer, performs CRC32, copies the cell header from

the cell header entry of the TCT, and sends the complete cell through the UTOPIA interface. For the last cell of an AAL5 frame, the transmitter appends the trailer of the common part conversion sublayer-protocol data unit (CPCS-PDU) to the user frame. It pads as required, appends the length (calculated during the frame transmit), and copies the CPCS-UU and CPI from the TxBD. The transmitter also sets the PTI[1] bit in the header. An interrupt can be optionally generated to declare the end of the transmit frame.

For AAL0, the transmitter simply copies the cell (except the HEC) prepared by the user from the channel's buffer and sends it through the UTOPIA interface. The ATM controller can optionally generate CRC10 on the cell payload and place the result at the end of the payload (CRC10 field). This feature is used to support OAM CRC10; refer to the ITU specification I.610 for additional details.

If, however, the current active channel's buffer is not ready, the transmit process ends and no cell is sent to the PHY. The PHY is responsible for generating an idle cell in an empty cell slot. An empty cell slot will continue to be generated each time the APC schedules this channel in the transmit queue until either a buffer is made ready or a TRANSMIT DEACTIVATE CHANNEL command is issued. See Section 38.3, "ATM Commands," for additional information about ATM controller commands.

NOTE:

The ATM controller does not generate the HEC in UTOPIA mode. The transmitter sends a dummy byte value (0x00) in place of the HEC; the PHY is responsible for the actual calculation of the HEC.

35.6.2 UTOPIA Receive Overview

The UTOPIA receive process begins when the PHY asserts the receive cell available signal (RxClav), indicating that the PHY has a complete cell in its receive FIFO buffer. The ATM controller first receives the cell header through the UTOPIA interface. The receiver translates the header address (GFC/VPI/VCI/PTI) to a channel number using either a look-up table in dual-port RAM, address compression tables in external RAM, or an external content-addressable memory (CAM). A cell header that has no match is treated as an AAL0 cell and is passed to the global raw cell queue (usually defined by convention to be channel number 0). If the cell header is matched to an active channel, the payload (48 bytes) is copied to the current buffer, the CRC is calculated (optional for AAL0), and the RCT parameters are updated. If, however, the current buffer is not empty, a busy interrupt is optionally generated and the cell is discarded.

For AAL5, when a cell with an end-of-frame marker (indicated by PTI[1]) is received, the receiver separates the trailer of the CPCS-PDU from the user data. The pads are removed as required, the length field is checked against the length which was calculated during the frame receive, the CPCS-UU and CPI are copied to the RxBD, and the buffer is closed. An interrupt is optionally generated to declare the end of received frame. If a CRC or length

error occurs, it is marked in the RxBD and an interrupt is optionally generated.

For AAL0, the ATM controller copies the cell (except the HEC) from the UTOPIA interface to the channel's current buffer and optionally performs a CRC10 check on the cell payload. The CRC10 option is used to support OAM cell checking (by host software) according to the ITU specification I.610.

NOTE:

The received HEC is not checked by the ATM controller in UTOPIA mode; it is the responsibility of the PHY to check the HEC and discard cells with an incorrect HEC.

35.6.3 Expanded Cells

An option for supporting ATM cells larger than the standard 53 bytes (4-byte header, 1-byte HEC and 48-byte payload) is available when operating in UTOPIA mode.

The MPC855T also supports cells up to 65 bytes in length (referred to as expanded cells) that use extra header fields for internal information in switching applications. Expanded cells consist of an expanded header of 0 to 12 bytes, a cell header (4 bytes), and a payload (48 bytes); the HEC can be optionally included. Figure 35-2 shows the structure of an expanded cell.

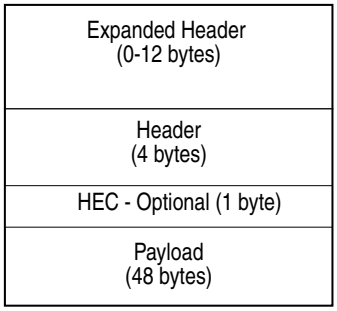


Figure 35-2. Expanded Cell Structure

For transmitting, the expanded header of each cell is taken from the expansion field in the TxBD and sent ahead of the cell header and payload. For receiving, the expanded header of the last cell of the current connection is copied to the expansion field in the connection's RxBD.

35.7 Serial ATM Operation

The SCC running serial ATM operates independently of the physical interface standard used. The TDM port can be used with serial ATM to allow an easy connection to an E1 or T1 line interface device. It is also possible to use the SCC directly through the NMSI.

In addition to the functions provided for UTOPIA operation, the serial ATM mode includes a transmission convergence (TC) layer. The TC layer provides cell delineation, scrambling, idle cell generation or filtering, and defines the interface characteristics to support E1/T1 or xDSL line interface devices.

35.7.1 Serial ATM Transmit Overview

The serial transmit process begins with the APC. The APC controls the ATM traffic of the transmitter through a user-configured timer that defines the maximum outgoing bit rate. The APC uses each ATM channel's specific traffic parameters to divide the total bit rate among the active channels. It can provide CBR and UBR traffic services. ABR can also be supported through software manipulation of APC parameters. The task of the APC is to define the next channel (or channels) to be transmitted. Refer to Chapter 39, "ATM Pace Control," for additional information about the operation and programming of the APC.

When operating in serial mode, transmit requests are internally generated by the SCC. The transmitter takes the next channel from the transmit queue for the SCC. It then reads the channel-specific information from the transmit connection table (TCT), and updates the TCT. The cell data is then copied from the transmit buffer to an internal buffer where the CRC32 and HEC are calculated, the cell header is appended, and scrambling is optionally performed. After the cell assembly process, the cell is moved into the SCC's transmit FIFO for transmission.

The transmitter appends the trailer of the CPCS-PDU in the last cell of an AAL5 user frame. The CPCS-PDU consists of the frame length (which is calculated during the frame transmit), the CPCS-UU and CPI fields from the TxBD, and cell padding as required. The transmitter also sets PTI[1] in the header of the last cell of the frame. An interrupt can be optionally generated to declare the end of a transmit frame.

When transmitting AAL0 cells, the ATM controller copies the cell (except the HEC) from the channel's cell transmit buffer. The ATM controller optionally generates CRC10 for the cell payload and places the result at the end of the payload. This feature is used to support OAM CRC10 per ITU specification I.610.

In the event of an empty transmit queue or no valid BDs for the active channel in the BD table, the transmitter generates an idle or unassigned cell (with the cell contents defined by the user). The transmitter sends idle cells if there are no channel numbers in the transmit queue or if the current channel in the transmit queue has no valid buffers. Channels can be removed from the transmit queue with the deactivate channel command. For additional information about ATM controller commands, refer to Section 38.3, "ATM Commands."

35.7.2 Serial ATM Receive Overview

The serial receive process starts after the receiver becomes synchronized with the incoming cells and can perform cell delineation. A receive request is then generated by theSCC. The receiver copies the first word from the SCC to the dual-port RAM (DPR). The receiver translates the header address (VCI/VPI/PTI) to a channel number through either a look-up table in dual-port RAM, address compression tables in external RAM, or an external content-addressable memory (CAM). If the header has no match the incoming cell is treated as an AAL0 cell, and is passed to the global raw cell queue (channel 0). If the cell is matched to a channel the channel status is read from the receive connection table (RCT). As the FIFO of the SCC fills, the received cell is read from the FIFO, the HEC is checked, and the cell is optionally descrambled. Cells with HEC errors are passed to the global raw cell queue, and the HEC error is recorded in the BD. The receiver screens out either idle cells or unassigned cells as programmed.

After each cell is assembled, either the entire cell (for AAL0 connections) or the cell payload (for AAL5 connections) is copied to external memory using SDMA channels, and the RCT is updated. If no empty buffer is available for the received channel in the BD table, an interrupt is generated and the cell is discarded.

CRC32 is checked on the cell payload for AAL5 connections, with pass/fail indication provided in the last BD of the received CPCS_PDU. The end of an AAL5 frame is indicated by the PTI[1] bit in the received cell header. When an end-of-frame indication occurs, the receiver separates the trailer of the CPCS-PDU from the user data. The length field is compared against the length calculated during the frame receive operation, the pads are removed as required, the CPCS-UU and CPI are copied to the BD, and the receive buffer is closed. An interrupt can be optionally generated to declare the end of a receive frame. Detected CRC or length errors are marked in the BD and an interrupt can be generated.

When AAL0 cells are received the ATM controller copies the cell (except the HEC) from the SCC FIFO to the next receive buffer in the channel’s BD table. The ATM controller calculates and checks CRC10 on the cell payload. This option supports the OAM cell check per ITU specification I.610.

35.7.2.1 Cell Delineation

In serial mode cell delineation is part of the receiver flow control. The ATM controller provides SDH/PDH oriented cell delineation on an octet basis using the HEC mechanism defined in the ITU specification I.432.

When cell reception begins, the ATM controller attempts to acquire the correct cell delineation. Once locked onto the cell boundaries of the incoming data stream, the ATM controller remains synchronized unless excessive errors disrupt the flow. A status bit (ASTATUS[LOCK]) indicates the current delineation status, and an interrupt (SCCE[SYNC]) can be generated whenever the cell lock status changes. Cells received before proper cell delineation is achieved are stored in the global raw cell queue.

35.7.3 Cell Payload Scrambling/Descrambling

Cell payload scrambling and descrambling can be performed in the cell stream using the $X^{43}+1$ scrambling algorithm. The ATM controller automatically transmits an empty cell following initialization to establish the 43-bit delay line and thereby avoid data corruption. On cell reception, the descrambling algorithm self-synchronizes before the HEC delineation process is complete and valid cell reception begins.

35.8 ATM Pace Control (APC)

The ATM pace controller determines the next channel (or *n* channels) to be transmitted and writes the channel number of these channels in the transmit queue every APC slot time. The transmitter sends one cell for each channel entry in the transmit queue.

Controlled by the communications processor (CP), the APC provides traffic shaping. Thus, the APC can flexibly combine SAR traffic with PTP traffic (cell switching). Its pace controller is based on multiple-level circular tables (APC scheduling tables) in the dual-port RAM that are used to schedule transmission of all the active channels.

The operation of the APC is controlled by several input parameters programmed by the user. Scheduling of traffic is controlled through the APC scheduling table length, the number of cells to be selected in an APC slot time, and the APC request timer. The period of the APC request timer determines the length of an APC slot time. The APC uses the APC_period parameter in the TCT to schedule the channel in the APC scheduling table.

For ABR transmission, the host adjusts the APC parameters in response to incoming resource management (RM) cells and defines the ABR available cell rate (ACR). The APC period can be changed on-the-fly, thereby allowing the bit rate for a channel to be changed dynamically, which is necessary to control transmission of traffic types such as ABR. For ABR, it is the user's responsibility to evaluate RM cells and update the APC_period entry in the TCT.

The APC input parameters are described in Chapter 39, "ATM Pace Control." These parameters define the minimum and maximum cell rate and cell delay variation.

35.9 Internal and External Channels (Extended Channel Mode)

Internal channels are the channels numbered 0 through 31; external channels have channel numbers greater than 31. The external channels become available only when the user selects extended channel mode (see Section 37.2, "SAR Receive State Register (SRSTATE)").

The TCTs and RCTs for internal channels are directly accessed in the (internal) dual-port RAM. For external channels, the TCT and RCT structures are placed in external memory

and thus require DMA accesses to read and update. Also, the GFC/VPI/VCI/PTI mapping for external channels requires a CAM or address compression method instead of the generally faster dual-port RAM look-up table method used for internal channels. Therefore, the bit rate supported in extended channel mode is reduced. The overall throughput depends on the number of external channels and the bit rate ratio between external and internal channels; that is, higher bit rate channels should be assigned internal channel numbers.

Chapter 36
Buffer Descriptors and Connection Tables

The communications processor module (CPM) manages ATM traffic through the UTOPIA and serial interfaces by means of transmit and receive buffer descriptors (BDs) and transmit and receive connection tables (TCTs and RCTs). The BDs are grouped into circular tables of pointers into the data buffer space in external memory. The following sections describe the structure and configuration of the BDs, TCTs, and RCTs.

36.1 ATM Buffer Descriptors (BDs)

ATM segmentation and reassembly (SAR) operates as a multi-channel protocol, segmenting and reassembling each channel's transmit and receive data to and from different sets of memory buffers simultaneously. The buffer descriptor (BD) implementation for ATM operation builds on the traditional SCC buffering method in which each controller has one pair of BD tables for receiving and transmitting. To accommodate multiple ATM channels, each channel number is given its own pair of BD tables located in external memory. The base pointers to a channel's BD tables are programmed as part of the channel-specific information in the channel's RCT and TCT; see Section 36.2, "Receive and Transmit Connection Tables (RCTs and TCTs)."

Each transmit channel has a separate TxBD table and a TCT which holds the TxBD pointers (TBASE and TBD_PTR). Likewise, each receive channel has a separate RxBD table and a RCT holding the RxBD pointers (RBASE and RBD_PTR). The global parameters TBDBASE and RBDBASE, located in the ATM-specific SCC (serial and UTOPIA) parameter RAM (see Chapter 37, "ATM Parameter RAM"), define the base addresses for the blocks of external memory containing all the channels' BD tables. The BD base offset pointers (TBASE and RBASE) point to the beginning of a given channel's BD tables. The BD offset pointers (TBD_PTR and RBD_PTR) specify the next (or currently active) BD.

Figure 36-1 shows TxBD tables and buffers and their associated pointers for two example transmit channels, channel 1 and channel 4. (The RxBD tables and buffers for receive channels have the same structure.)

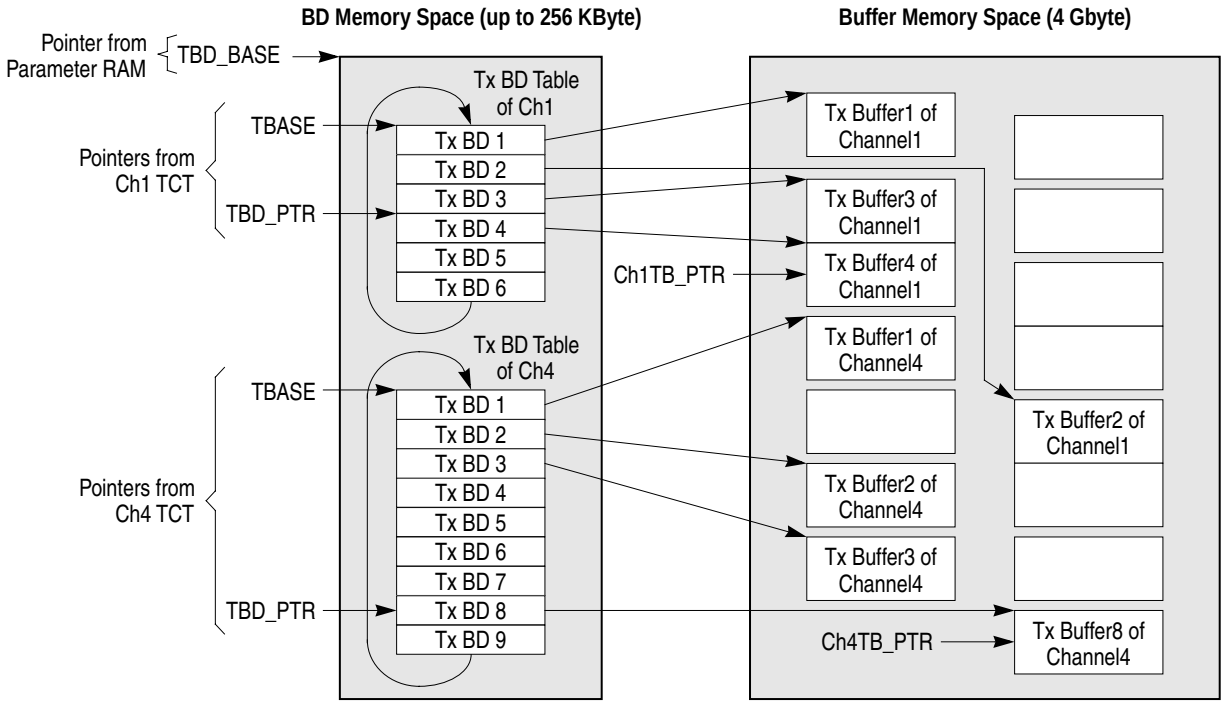


Figure 36-1. Transmit Buffer and TxBD Table Example

In the example of Figure 36-1, when the transmitter encounters channel number 1 in the transmit queue, it sends a cell using transmit buffer 4 because TxBD 4 is the active BD in channel number 1's TxBD table. As the cell is sent, the transmit buffer pointer TB_PTR tracks the current data position. After the cell is sent, the transmitter advances the channel's TBD_PTR to the next BD in the table (unless the current buffer is a multiple-cell AAL5 frame in which case TBD_PTR is not advanced and the current data position is kept in TB_PTR) and moves on to the next channel in the transmit queue. When the end of the BD table is reached (TxBD[W] = 1), the transmitter returns to the head of the table by re-initializing TBD_PTR to the channel's TBASE address.

36.1.1 AAL5 Buffers

Each AAL5 buffer can hold either a whole frame or part of it. During transmit or receive operations, interrupts are optionally generated at the closing of each buffer or at the end of a frame.

The last buffer of a frame is padded automatically by the transmitter to fit an AAL5 cell payload according to ITU specification I.363. The transmit buffer data length for AAL5 transmit buffers must be greater than or equal to 48 bytes for all buffers except the first and last buffer of a frame; the first and last buffers of a frame must have a data length greater than zero. Note that AAL5 transmit buffers have no alignment restrictions.

Receive buffers, however, must start on a burst-aligned address (divisible by 16) and their lengths should be a multiple of 48 bytes (that is, the value of SMRBLR in the SCC

parameter RAM should be a multiple of 48). The buffers are filled with multiples of 48 bytes, except for the last buffer in a frame from which the AAL5 pads are removed.

36.1.2 AAL0 Buffers

AAL0 buffers contain one raw cell. When the receiver or transmitter completes writing or reading the buffer, it moves to the next BD in the AAL0 channel's BD table in preparation for the next transfer and optionally issues an interrupt.

AAL0 buffers are 64 bytes. 52 bytes are used to hold the cell header and payload—the HEC is not included in the receive or transmit buffers. The remaining 12 bytes of the buffer are not used and are available to the user. The AAL0 buffer structure is shown in Figure 36-2.

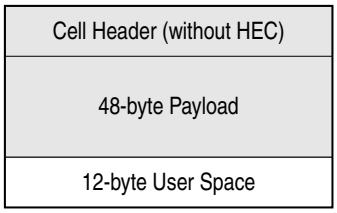


Figure 36-2. AAL0 Buffer Structure

Unlike other protocols, both the AAL0 transmit and receive buffers should be 16-byte aligned.

36.1.3 ATM Receive Buffer Descriptors (RxBDs)

The format of the ATM receive buffer descriptor (RxBD) applies to both UTOPIA and serial ATM modes. ATM RxBDs are 12 bytes, as shown in Figure 36-3.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	E	—	W	I	L	F	CM	—	HEC	CLP	CNG	ABT	—	—	LN	CR
OFFSET + 2	DATA LENGTH/*CHANNEL Code*															
OFFSET + 4	RX DATA BUFFER POINTER															
OFFSET + 8	CPCS-UU+CPI															
OFFSET + A	—															

Figure 36-3. ATM RxBD

For UTOPIA operation, a global option to support expanded cells is available. ATM RxBDs in expanded cell mode are 24 bytes, as shown in Figure 36-4.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	E	—	W	I	L	F	CM	—	HEC	CLP	CNG	ABT	—	—	LN	CR
OFFSET + 2	DATA LENGTH/*CHANNEL Code*															
OFFSET + 4	RX DATA BUFFER POINTER															
OFFSET + 8	CPCS-UU+CPI															
OFFSET + A	—															
OFFSET + C	Cell Header Expansion 1															
OFFSET + 10	Cell Header Expansion 2															
OFFSET + 14	Cell Header Expansion 3															

Figure 36-4. ATM RxBD in Expanded Cell Mode (UTOPIA Only)

Table 36-1 describes the ATM RxBD fields.

Table 36-1. ATM RxBD Field Descriptions

Offset from RBD_PTR	Bits	Name	Description
0x00	0	E	Empty. Determines whether a buffer is accessible by the CPU core or the CP. 0 The data buffer associated with this RxBD has been filled with the received data, or data reception has been aborted due to an error condition. The CPU core is free to examine or write to this RxBD and buffer. The CP will not access a BD while the E-bit is cleared. 1 The data buffer associated with this RxBD is empty, or reception is currently in progress. This RxBD and its associated receive buffer are in use by the CP. Once the E bit is set, the CPU core should not write to this RxBD.
	1	—	Reserved
	2	W	Wrap. Determines that this is the final BD in the table. 0 This is not the last BD in the RxBD table. 1 This is the last BD in the RxBD table. The next time this channel receives data, the first BD in the channel's RxBD table (the BD pointed to by the channels's RCT[RBASE] address) will be used. The number of RxBDs in this table is programmable and is determined only by the W bit and the overall space constraints of the 256K memory space available for all the receive channels.
	3	I	Interrupt. Enables RXB interrupts when a receive buffer is closed during receipt of an AAL5 frame buffer or an AAL0 buffer. Note that RXF (receive AAL5 frame) interrupts are not affected by this bit. 0 No interrupt is generated after this buffer has been used. 1 Interrupt occurs after this buffer has been closed by the ATM controller. This class of interrupt is indicated through the setting of the RXB bit in an entry in the interrupt queue.

Table 36-1. ATM RxBD Field Descriptions (continued)

Offset from RBD_PTR	Bits	Name	Description
0x00	4	L	Last in frame (AAL5). Set by the ATM controller when this buffer is the last in an AAL5 frame or when an error occurs, in which case one or more error bits are also set. The ATM controller writes the total number of frame octets to the data length field. 0 The buffer is not the last in a frame. 1 The buffer is the last in a frame or an error has occurred.
	5	F	F (AAL5) First in frame. Set by the ATM controller when this buffer is the first in a frame 0 The buffer is not the first in a frame. 1 The buffer is the first in a frame.
	6	CM	Continuous mode. Note that RxBD[E] is cleared if an error occurs during reception, regardless of CM. 0 Normal operation. 1 RxBD[E] is not cleared by the CPM after this BD is closed, allowing the associated buffer to be overwritten next time the CPM accesses it.
	7	—	Reserved.
	8	HEC	HEC error (AAL5). A receiver HEC error occurred on at least one cell of the frame. Cells with HEC errors are passed to the global raw cell queue with this bit set.
	9	CLP	Cell loss priority. Indicates that at least one cell was received with its CLP bit set. This bit is set in the last BD in the frame by the CP for channels that implement AAL5. For channels that implement AAL0, this bit is copied from the CLP bit in the header of the received cell.
	10	CNG	Congestion indication. Indicates that the last cell in the frame was received with its PTI congestion bit set. This bit is set in the last BD of the frame by the CP for channels that implement AAL5. For channels that implement AAL0, this bit is copied from the PTI bit in the header of the received cell.
	11	ABT	Abort (AAL5). Indicates that a frame abort was detected (last cell in frame indicated zero frame length.) This error bit is set in the last BD of the aborted frame by the CP for channels that implement AAL5.
	12–13	—	Reserved
	14	LN	Receive length error (AAL5). Indicates the number of octets received in the frame does not match the length specified in the length field of the AAL5 frame. This bit is set in the last BD of the frame by the CP for channels that implement AAL5. Note that the whole received PDU is written to the data buffer even if a receive length error is detected.
	15	CR	Receive CRC error. Indicates a CRC32 error for AAL5 channels or a CRC10 error for AAL0 channels. The receiver performs CRC32 checking on the frame for AAL5 channels and sets the CR bit in the last BD when a CRC error is detected. For AAL0 channels, the receiver performs CRC10 checking on the cell payload and sets the CR bit if a CRC error is detected. 0 No error 1 CRC32 error for AAL5 channels; CRC10 failure for AAL0 channels Note: RxBD[CR] is set only when the received cell includes a CRC10 field in the payload and an error is detected. This check is provided to support OAM CRC10 according to the ITU specification I.610.

Table 36-1. ATM RxBD Field Descriptions (continued)

Offset from RBD_PTR	Bits	Name	Description
0x02	—	Data length	Data length. For AAL5 BDs, the data length is the number of octets written by the CP into the BD's data buffer. When the buffer is the last buffer in the frame (BD[L] = 1), the data length contains the total number of frame octets. This field is written by the CP as the buffer is closed.
0x04	—	Receive data buffer pointer	Receive data buffer pointer. Points to the first location of this BD's data buffer, which may reside in either internal or external memory. This pointer must be burst aligned (divisible by 16).
0x08	—	CPCS-UU and CPI	CPCS-UU and CPI (AAL5 only). Contains the frame's CPCS-UU and CPI fields. This field is taken from the frame trailer, and contains user-to-user (UU) information and common part indications (CPI). This field is written by the CP and is valid only for the last BD in the AAL5 frame.
0x0A	—	—	Reserved, should be cleared.
0x0C	—	Cell header expansion 1, 2, and 3	Cell header expansion 1, 2, and 3 (UTOPIA expanded cell mode only). These fields are added to the BD when expanded cells are enabled (SRSTATE[EC] is set in the UTOPIA parameter RAM). The size of the cell header expansion is programmable and is determined by ECSIZE in the UTOPIA parameter RAM. When the last BD of an AAL5 frame is closed, the ATM controller copies the expanded cell header into these fields. For AAL0 channels, the expanded cell headers are copied into these fields for every RxBD. Cell header expansion 1 is the first word to be received; cell header expansion 3 is the last. The bytes are ordered according to the DMA byte ordering (programmed through SRFCR in the parameter RAM).

36.1.4 ATM Transmit Buffer Descriptors (TxBDs)

The format of the ATM transmit buffer descriptor (TxBD) applies to both UTOPIA and serial ATM modes. ATM TxBDs are 12 bytes, as shown in Figure 36-5.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	R	—	W	I	L	—	CM	—	—	—	—	—	—	—	—	—
OFFSET + 2	DATA LENGTH															
OFFSET + 4	TX DATA BUFFER POINTER															
OFFSET + 8	CPCS-UU+CPI/*HEADER_L*															
OFFSET + A	Reserved/*HEADER_H*															

Figure 36-5. ATM TxBD

For UTOPIA operation, a global option to support expanded cells is available. ATM TxBDs in expanded cell mode are 24 bytes, as shown in Figure 36-6.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OFFSET + 0	R	—	W	I	L	—	CM	—	—	—	—	—	—	—	—	—
OFFSET + 2	DATA LENGTH															
OFFSET + 4	TX DATA BUFFER POINTER															
OFFSET + 8	CPCS-UU+CPI/*HEADER_L*															
OFFSET + A	Reserved/*HEADER_H*															
OFFSET + C	Cell Header Expansion 1															
OFFSET + 10	Cell Header Expansion 2															
OFFSET + 14	Cell Header Expansion 3															

Figure 36-6. ATM TxBD in Expanded Cell Mode (UTOPIA Only)

Table 36-2 describes the ATM TxBD fields.

Table 36-2. ATM TxBD Field Descriptions

Offset from TBD_PTR	Bits	Name	Description
0x00	0	R	Ready. Determines whether the data buffer is ready for transmission. 0 The data buffer associated with this BD is not ready for transmission. The CPU core is free to manipulate this BD and its associated data buffer. The CP clears this bit after the buffer has been sent or after an error condition is encountered. 1 The data buffer, which has been prepared for transmission by the user, has not been sent or is currently being sent. This BD and its buffer should not be modified by the CPU core.
	1	—	Reserved
	2	W	Wrap. Determines whether this is the final BD in the table. 0 This is not the last BD in the TxBD table. 1 This is the last BD in the TxBD table. The next time this channel is scheduled to send data, the first BD in the channel's TxBD table (the BD pointed to by the channels's TCT[TBASE] address) will be used. The number of TxBDs in the table is programmable and is limited by the 256K memory space for all the transmit channels.
	3	I	Interrupt. Enables interrupts generated when the contents of the buffer have been sent. 0 No interrupt is generated after the buffer has been transmitted. 1 The TXB bit is set in an entry in the interrupt queue after the buffer has been sent.
	4	L	Last in frame (AAL5). Set by the user for the last buffer in an AAL5 frame. 0 This is not the last buffer in the transmit frame. 1 This is the last buffer in the current transmit frame.
	5	—	Reserved
	6	CM	Continuous mode. Note that TxBD[R] is cleared if an error occurs during transmission, regardless of CM. 0 Normal operation. 1 The CP does not clear TxBD[R] after this BD is closed, allowing the buffer to be resent the next time the CP accesses this BD.
	7–15	—	Reserved

Table 36-2. ATM TxBD Field Descriptions (continued)

Offset from TBD_PTR	Bits	Name	Description
0x02	—	Data length	Specifies the number of octets the ATM controller sends from this BD's data buffer. The value of data length should follow the guidelines in Section 36.1.1, "AAL5 Buffers," or Section 36.1.2, "AAL0 Buffers," as appropriate. This value is not modified by the CP.
0x04	—	Transmit data buffer pointer	Contains the address of the associated data buffer. The buffer may reside in either internal or external memory. This value is not modified by the CP.
0x08	—	CPCS-UU and CPI	CPCS-UU and CPI (AAL5 only and RH = 0 and L = 1). Valid only when the current BD is the last BD of an AAL5 frame. CPCS-UU and CPI are used in the frame trailer to transfer user-to-user (UU) information and common part indications (CPI). The transmitter copies this field to the AAL5 frame trailer.
0x0C	—	Cell header expansion 1, 2, and 3	Cell header expansion 1, 2, and 3 (UTOPIA expanded cell mode only). These fields are added to the BD when expanded cells are enabled (SRSTATE[EC] is set in the UTOPIA parameter RAM). The size of the cell header expansion is programmable and is determined by ECSIZE in the UTOPIA parameter RAM. For AAL5 channels, the ATM controller copies the contents of the cell header expansion fields from the first BD of the current frame and appends them to every cell of the frame sent. For AAL0 channels, these fields are copied into the expanded cell header of the cell. Cell header expansion 1 is the first word to be sent; cell header expansion 3 is the last. The bytes are ordered according to the DMA byte ordering (programmed through STFCR in the parameter RAM).

36.2 Receive and Transmit Connection Tables (RCTs and TCTs)

The receive and transmit connection tables (RCTs and TCTs) hold configuration and control information and temporary parameters for each receive and transmit ATM channel. Although the transmit and receive sections of the same channel number are independent of each other, their RCTs and TCTs are paired together in connection tables (CTs). Each CT is 64 bytes. The upper 32 bytes is used for the RCT, and the lower 32 bytes is the TCT.

The CTs for internal channels (channels 0–31) are in the dual-port RAM. In extended channel mode (UTOPIA only), the tables for external channels (numbered 32 and above) are kept in external memory. The structure of the CTs is shown in Figure 36-7.

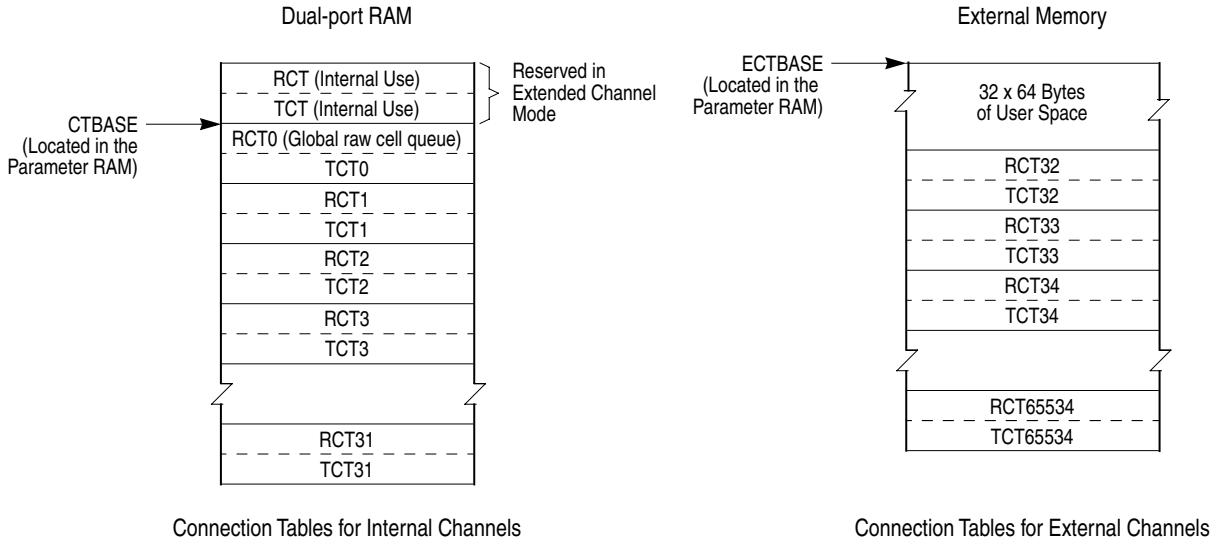


Figure 36-7. Connection Tables in Dual-port RAM and External Memory

Note that some CT space is reserved. RCT0 is reserved for the global raw cell queue. (TCT0 is still available to the user.) In extended channel mode, an additional CT pair immediately above the CTBASE pointer is reserved for internal use.

36.2.1 Receive Connection Table (RCT)

Each receive connection table (RCT) holds parameters (channel configuration, pointers, status flags, and temporary data) for a single ATM receive channel. Figure 36-8 shows the RCT structure.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CT_Offset + 0	FHNT	—	—	—	—	—	—	—	HEC	CLP	CNG	INF	CNGI	CDIS	AAL	
CT_Offset + 2	RBALEN															
CT_Offset + 4	RCRC															
CT_Offset + 6																
CT_Offset + 8	RB_PTR															
CT_Offset + A																
CT_Offset + C	RTMLEN															
CT_Offset + E	RBD_PTR															
CT_Offset + 10	RBASE															
CT_Offset + 12	TSTAMP															
CT_Offset + 14	IMASK															
CT_Offset + 16 to CT_Offset + 1F	—															

Figure 36-8. Receive Connection Table (RCT)

Table 36-3 describes the RCT fields.

Table 36-3. RCT Field Descriptions

CT Offset	Bits	Name	Description
0x00	0	FHNT	Frame hunt mode. Indicates that this channel has had a busy exception or a restart and is currently in frame hunt mode. In frame hunt mode the CP discards all received cells until a new frame is started (indicated by the CPI bit of the last cell header). FHNT is cleared by the user during initialization, and is modified accordingly by the CP thereafter. 0 Not in frame hunt mode 1 In frame hunt mode
	1-7	—	Reserved, should be cleared.
	8	HEC	Header error control indication (serial ATM only). Used internally by the CP, RCT[HEC] indicates that a HEC error has been detected in at least one cell of the current frame. For AAL5, this bit is set as soon as a cell of the current frame is determined to have a HEC error. At the end of the frame, this bit is copied to the last RxBD[HEC] of the frame. For AAL0, this bit is copied to the RxBD[HEC] for each cell. 0 No HEC error has been detected. 1 A HEC error has been detected.
	9	CLP	Cell loss priority indication (AAL5 only). Used internally by the CP, RCT[CLP] indicates that at least one low priority cell has arrived as part of the current AAL5 frame. This bit is set as soon as a cell of the current frame is determined to have a CLP bit set in its header. At the end of the frame, this bit is copied to the last RxBD[CLP] of the frame. For AAL0 channels, the CLP indication is available in the cell header in the data buffer. 0 No low priority cells have arrived in the current frame. 1 A low priority cell has arrived during the current frame.

Table 36-3. RCT Field Descriptions (continued)

CT Offset	Bits	Name	Description
0x00	10	CNG	CNG—Congestion (AAL5 only). Used internally by the CP, RCT[CNG] indicates that congestion has been reported in the last cell of the current frame. RCT[CNG] is set only if the last cell of the current AAL5 frame has arrived with the PTI[EFCI] bit set in its header. At the end of the frame, this bit is copied to the last RxBD[CNG] of the frame. 0 No congestion has been reported. 1 Congestion has been reported.
	11	INF	In frame. The receiver sets RCT[INF] when a new frame starts and clears it when the frame ends. This bit is used internally. 0 Idle state. No buffer is open for this channel. 1 Receiving. The receiver is in the middle of a frame; a receive buffer is open.
	12	CNGI	CNGI (AAL5 only)- Congestion interrupt. This bit enables interrupts when the SAR layer detects a cell's PTI[EFCI] bit is set. 0 Disable congestion interrupts. Congestion is indicated in the last RxBD[CNG] of the frame. 1 Enable congestion interrupts. An interrupt entry is added to the interrupt queue if PTI[EFCI] is set in a received cell. See Chapter 40, "ATM Exceptions."
	13	CDIS	Channel disable status. Set by the STOP RECEIVE command; cleared by the RESTART RECEIVE command. 0 The channel is enabled to receive cells. 1 The channel is disabled. Cells addressed to this channel are discarded. Note that the user should not modify this bit directly; that is, the STOP RECEIVE and RESTART RECEIVE commands should be used to control the CDIS bit. However, CDIS should be cleared during initialization.
	14–15	AAL	AAL type. Selects the ATM adaptation layer support for receive cells. 00 AAL0 (raw-cell channel). The receiver stores cells using the AAL0 buffer format. 01 AAL5. The receiver performs AAL5 cell reassembly and copies the 48 bytes of payload to the receive buffer. The AAL5 frame trailer is checked when the last cell of the PDU is received. 1x Reserved
0x02	—	RBALEN	(AAL5 only) Receive buffer available length. Contains the number of bytes available in the receive buffer. When a buffer is opened, the CP initializes RBALEN with the value programmed in SMRBLR in parameter RAM. RBALEN is then decremented by 48 for each cell received by the channel.
0x04	—	RCRC	(AAL5 only) Receive CRC. Contains the CRC32 value calculated during a cell receive operation.
0x08	—	RB_PTR	Receive buffer pointer. This field is valid only when INF is set. RB_PTR is the physical address of the current buffer location to which data is being written. Should be cleared during initialization.
0x0C	—	RTMLEN	(AAL5 only) Frame buffer count. Contains the total number of bytes received during the current AAL5 frame. The CP clears RTMLEN at the beginning of a frame and increments it by the value in SMRBLR in parameter RAM as each additional buffer is received. The receiver uses RTMLEN to calculate the total frame length which is then compared to the length field of the frame.

Table 36-3. RCT Field Descriptions (continued)

CT Offset	Bits	Name	Description
0x0E	—	RBD_PTR	RxBD pointer. Points to the current BD in the RxBD table. The actual address of the current BD is (RBD_PTR x 4) + RBDBASE (where RBDBASE is the base pointer to the RxBD memory space). Initialize RBD_PTR to the same value as RBASE. Note that RBD_PTR is a word-aligned offset pointer from RBDBASE; that is, it provides bits [14–29] of the offset, and bits [30–31] are always 00.
0x10	—	RBASE	RxBD table base. Points to the first BD in the RxBD table of this channel. The actual address of the table is (RBASE x 4) plus the base address RBDBASE in the parameter RAM; see. Note that RBASE is a word-aligned offset pointer from RBDBASE; that is, it provides bits [14–29] of the offset, and bits [30–31] are always 00.
0x12	—	TSTAMP	Time stamp. Contains the arrival time stamp for the current frame. The CP copies the time value taken from the CP timer (selected by TSTA in parameter RAM) to the TSTAMP field on the arrival of the frame's first cell. TSTAMP can be used to check for a time-out condition.
0x14	—	IMASK	Interrupt mask. Contains the interrupt mask for both the receive and transmit sides of this channel number. The interrupt mask allows the user to enable or disable interrupt generation. If a bit in IMASK is cleared, interrupt queue entries are not generated for the corresponding event, and the GINT global interrupt counter is not advanced. See Section 40.3, "Interrupt Queue Mask (IMASK)."
0x16–0x1F	—	—	Reserved, should be cleared.

36.2.2 Transmit Connection Table (TCT)

Each TCT holds parameters (channel configuration, pointers, status flags, and temporary data) for a single ATM transmit channel. Figure 36-9 shows the TCT structure.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CT_Offset + 20	—	—	—	—	—	—	—	—	PC	—	—	INF	CR10	CDIS	AAL	
CT_Offset + 22	TBALEN															
CT_Offset + 24	TCRC															
CT_Offset + 28	TB_PTR															
CT_Offset + 2C	TTMLEN															
CT_Offset + 2E	TBD_PTR															
CT_Offset + 30	TBASE															
CT_Offset + 32	—															
CT_Offset + 34	CHEAD															
CT_Offset + 38	APCL															
CT_Offset + 3A	APCPR															
CT_Offset + 3C	OUT	APCP														
CT_Offset + 3E	APCPF															

Figure 36-9. Transmit Connection Table (TCT)

Table 36-4 describes the TCT fields.

Table 36-4. TCT Field Descriptions

CT Offset	Bits	Name	Description
0x20	0–7	—	Reserved
	8	PC	Padded cell. Indicates that the transmitter must add an additional padded cell to the end of an AAL5 frame. The user must clear this bit during initialization.
	9–10	—	Reserved
	11	INF	In frame status. The transmitter sets TCT[INF] when a new frame starts and clears it when the frame ends. This bit is used internally. 0 Idle state. No buffer is open for this channel. 1 Sending. The transmitter is in the middle of a frame; a transmit buffer is open.
	12	CR10	Perform CRC10 (AAL0 only). This bit enables CRC10 calculation on the transmit cell payload for OAM support according to the ITU specification I.610. 0 Do not perform CRC10. 1 Perform CRC10 and insert the result into the CRC field (the last 10 bits of the OAM cell payload).
	13	CDIS	Channel disable status. Set by the STOP TRANSMIT (abort) command; cleared by the RESTART TRANSMIT command. 0 Transmission is enabled. 1 Transmission is disabled. Note that the user should not modify this bit directly; that is, the STOP TRANSMIT and RESTART TRANSMIT commands should be used to control the CDIS bit. However, CDIS should be cleared during initialization.

Table 36-4. TCT Field Descriptions (continued)

CT Offset	Bits	Name	Description
0x20	14–15	AAL	AAL type. Selects the ATM adaptation layer support for transmit cells. 00 AAL0 (raw-cell channel). The transmitter sends cells using the AAL0 buffer format. 01 AAL5. The transmitter performs AAL5 cell segmentation and sends the 48 bytes of payload from the transmit buffer. The AAL5 frame trailer is generated when the last cell of the PDU is sent. 1x Reserved
0x22	—	TBALEN	Transmit buffer available length (AAL5 only). Contains the number of bytes in the current transmit buffer. TBALEN is initialized with the data length field of the TxBD when a new buffer opens and is decremented by 48 for each cell sent.
0x24	—	TCRC	Temporary CRC32 (AAL5 only). CP scratch pad area for the CRC32 calculation.
0x28	—	TB_PTR	Transmit buffer pointer. Contains the real address of the current data position in the transmit buffer.
0x2C	—	TTMLEN	Transmit total message length (AAL5 only). Counts bytes sent in the current frame. The transmitter initializes TTMLEN with the data length field of a frame's first TxBD. TTMLEN accumulates the buffer data lengths as each additional BD is opened. At the end of the frame, TTMLEN is copied into the AAL5 length field.
0x2E	—	TBD_PTR	TxBD pointer. Points to the current BD in the TxBD table. The actual address of the current BD is (TBD_PTR x 4) + TBDBASE (where TBDBASE is the base pointer to the TxBD memory space). Initialize TBD_PTR to the same value as TBASE. Note that TBD_PTR is a word-aligned offset pointer from TBDBASE; that is, it provides bits [14–29] of the offset, and bits [30–31] are always 00.
0x30	—	TBASE	TxBD table base. Points to the first BD in the TxBD table of this channel. The actual address of the first BD in the table is (TBASE x 4) plus the offset value TBDBASE in parameter RAM; see. Note that TBASE is a word-aligned offset pointer from TBDBASE; that is, it provides bits [14–29] of the offset, and bits [30–31] are always 00.
0x32	—	—	Reserved
0x34	—	CHEAD	Channel header (AAL5 only). Contains the full (4-byte) cell header for this AAL5 channel. The transmitter appends CHEAD and the calculated HEC to the payload to create a complete cell. This field should be initialized by the user and should not be modified while the channel is active. Note that the byte ordering of this field is reversed; that is, CHEAD[0–7] is the LSB with CHEAD[0] the msb, and CHEAD[24–31] is the MSB with CHEAD[24] the msb.
0x38	—	APCL	APC link. Used by the CP to link additional channels to the same APC time slot occupied by the current channel. Initialize this field with 0xFFFF to indicate the end of the linked list.
0x3A	—	APCPR	APC pace remainder. Contains the remainder of the rate generated by the APC after adding the pace FRACTION to the cumulative APCPR. Should be cleared during initialization.

Table 36-4. TCT Field Descriptions (continued)

CT Offset	Bits	Name	Description
0x3C	0	OUT	APC out. Can be used as a completion flag for the TRANSMIT DEACTIVATE CHANNEL command. When the TRANSMIT DEACTIVATE CHANNEL command is issued, OUT is immediately set. OUT is then cleared when the channel is actually removed from the APC scheduling table. 0 No TRANSMIT DEACTIVATE CHANNEL command is pending. 1 The channel is waiting to be removed from the APC scheduling table.
	1–15	APCP	APC pace. Contains the channel's APC pacing. When the channel is placed in the transmit queue, the APC reschedules the channel in a new position (time slot) in the APC scheduling table. The new slot position is (the current slot position + APCP) modulo the table length. Note that the APCP value selected is bounded as shown below: $1 \le \left(\mathrm{APCP} + \frac{\mathrm{APCPF}}{65536}\right) \le [\mathrm{APCTablelength} - 1]$ For more information about the APC, see Chapter 39, "ATM Pace Control."
0x3E	—	APCPF	APC pace fraction. Contains the channel's APCP_FRACTION in units of 1/65536 of a time slot. For example, a pace of 1.5 is obtained by programming APCP to 1 and APCPF to 0x8000. The pace thus becomes 1+32768/65536, or a value of 1.5.

Chapter 37
ATM Parameter RAM

The SCC parameter RAM is used to configure the SCC for serial ATM and the UTOPIA interface. The CP also uses parameter RAM to store operational and temporary values used during SAR activities.

When ATM operations are performed, the SCC parameter RAM is mapped as shown in Table 37-1, Table 37-2, and Table 37-2. The values written in the parameter RAM by the user or the CP determine the ATM capabilities of the SCC and the UTOPIA interface. Table 37-1 describes the shared parameters for serial ATM and UTOPIA modes. Note that the shaded table entries are for serial ATM only.

Table 37-1. Serial ATM and UTOPIA Interface Parameter RAM Map

Offset [1]	Name	Width	Description
0x00	RBDBASE	Word	Base pointer for all RxBD tables. Defines the starting location in external memory of up to 256 Kbytes in which the RxBD tables for all connections are located. The RxBD table pointer for a specific connection is offset from RBDBASE and is located in the channel's RCT[RBASE]. (For the receive channel of a PTP connection, the PTP BD table pointer is offset from RBDBASE and is located in the channel's PTP RCT[PTP_BASE].) RBDBASE should be word aligned.
0x04	SRFCR	Byte	SAR receive function code register. Contains global parameters for DMA transfers. See Section 37.1, "SAR Receive Function Code Register (SRFCR)."
0x05	SRSTATE	Byte	SAR receive state. Contains global state parameters. See Section 37.2, "SAR Receive State Register (SRSTATE)."
0x06	MRBLR	Hword	Maximum receive buffer length register. MRBLR should be cleared for ATM operation; that is, if MRBLR is programmed with a non-zero value the SCC operates in transparent mode. The SAR MRBLR (SMRBLR) field is used instead (see below).
0x08	RSTATE	Word	SCC internal receive state parameters. Stores internal state variables and flags. During initialization, copy the value of SRFCR into the MSB of RSTATE; the less-significant bytes should be cleared. Do not write to RSTATE during receive operations.
0x0C	—	Word	Reserved
0x10	R_CNT	Hword	Receive internal byte counter. Counts the bytes received during ATM cell reception. The user must not write to this location.
0x12	STFCR	Byte	SAR transmit function code register. Contains global parameters for DMA transfers. See Section 37.3, "SAR Transmit Function Code Register (STFCR)."

Table 37-1. Serial ATM and UTOPIA Interface Parameter RAM Map (continued)

Offset [1]	Name	Width	Description
0x13	STSTATE	Byte	SAR transmit state. Contains global state parameters. See Section 37.4, “SAR Transmit State Register (STSTATE).”
0x14	TBDBASE	Word	Base pointer for all TxBD tables. Defines the starting location in external memory of up to 256 Kbytes in which the TxBD tables for all connections are located. The TxBD table pointer for a specific connection is offset from TBDBASE and is located in the channel’s TCT[TBASE]. (For the transmit channel of a PTP connection, the PTP BD table pointer is offset from TBDBASE and is located in the channel’s PTP TCT[PTP_BASE].) TBDBASE should be word aligned.
0x18	TSTATE	Word	SCC internal transmit state parameters. Stores internal state variables and flags. During initialization, copy the value of STFCR into the MSB of TSTATE; the less-significant bytes should be cleared. Do not write to TSTATE during transmit operations.
0x1C	COMM_CH	Hword	Command channel. Contains the channel number associated with current channel command. The host should write the required channel number to COMM_CH before a channel-specific command is issued. See Section 38.3, “ATM Commands.”
0x1E	STCHNUM [2]	Hword	Current SAR transmit channel number (channel code). Internal use.
0x20	T_CNT	Hword	Transmit internal byte counter. Counts the bytes sent during ATM cell transmission. The user must not write to this location.
0x22	CTBASE	Hword	Connection table base address. Contains the 64-byte-aligned base address in the dual-port RAM for the connection tables of the internal channels (numbered 0–31). CTBASE is an offset from the beginning of dual-port RAM. See Section 36.2, “Receive and Transmit Connection Tables (RCTs and TCTs).”
0x24	ECTBASE	Word	External connection table base address. Valid only in extended channel mode (S*x*STATE[EXT] = 1). Contains the 64-byte-aligned base address for the connection tables of the external channels (numbered 32 and higher). See Section 36.2, “Receive and Transmit Connection Tables (RCTs and TCTs).”
0x28	INTBASE	Word	Interrupt base pointer. Contains the word-aligned starting address of the interrupt queue in external memory. See Chapter 40, “ATM Exceptions.”
0x2C	INTPTR	Word	Pointer to interrupt queue. Contains the address of the next interrupt entry in the interrupt queue. See Chapter 40, “ATM Exceptions.” Copy the value of INTBASE into INTPTR before enabling interrupts.
0x30	C-MASK	Word	Constant mask for CRC32. C-MASK is used by the receiver to check CRC32 results when supporting AAL5 connections. Initialize to 0xDEBB_20E3.
0x34	SRCHNUM[2]	Hword	Current SAR receive channel number (channel code). Internal use.
0x36	INT_CNT	Hword	Interrupt counter. Initialize with INT_ICNT. The CP decrements INT_CNT for each interrupt added to the interrupt queue. When INT_CNT reaches zero, the CP sets the queue’s global interrupt flag (SCCE[GINT] or IDSR1[GINT]) and reinitializes the counter with INT_ICNT. See also Chapter 40, “ATM Exceptions.”
0x38	INT_ICNT	Hword	Interrupt initial count. INT_ICNT is the user-defined global interrupt threshold—the number of interrupts required before the CP issues a global interrupt through SCCE[GINT] or IDSR1[GINT].

Table 37-1. Serial ATM and UTOPIA Interface Parameter RAM Map (continued)

Offset [1]	Name	Width	Description
0x3A	TSTA	Hword	Time stamp timer address (AAL5 only). Contains the address of the RISC timer to be used for the time-out process. The receiver copies the RISC timer value from the address specified by the TSTA field to the RCT when a new frame is received. TSTA should be initialized with a value equal to (TM_BASE + 4*timer number).
0x3C	OLDLEN[2]	Hword	Transmitter temporary length. Do not write to this location.
0x3E	SMRBLR	Hword	SAR maximum receive buffer length register. Determines the number of bytes the CP writes to a receive buffer before moving to the next buffer. SMRBLR is user-defined and should be a multiple of 48 bytes. SMRBLR is global for all ATM connections per controller.
0x40	EHEAD	Word	Empty cell header and empty cell payload for serial mode. Contain the data for the empty cell header (EHEAD) and payload (EPAYLOAD). The CP sends and receives empty (idle) cells using the data contained in EHEAD and EPAYLOAD. When transmitting, the CP assembles an empty cell by sending EHEAD once, calculating and sending a HEC, and then sending EPAYLOAD twelve times (48 bytes). When receiving, the CP compares the incoming header with EHEAD to check for empty cells; if they match the cell is discarded. The user should program EHEAD and EPAYLOAD as required: The ATM Forum UNI specification states that unassigned cells should be sent when no valid transmit data is available, while the ITU mandates the use of idle cells. Unassigned cells are used as empty cells when EHEAD = 0x0000_0000 and idle cells when EHEAD = 0x0100_0000. In both cases, EPAYLOAD should be initialized to 0x6A6A_6A6A. Note that the data for these fields must be written in little-endian byte order.
0x44	EPAYLOAD	Word	
0x48	TQBASE	Hword	Transmit queue base pointer. Contains the user-defined pointer to the base address of the transmit queue in the dual-port RAM. See Section 39.6, “PHY Transmit Queues.”
0x4A	TQEND	Hword	Transmit queue end pointer. Contains the user-defined pointer to the last entry in the transmit queue in the dual-port RAM. The size of the transmit queue is user-defined, but note that the minimum number of entries is (NCITS +2). See Section 39.6, “PHY Transmit Queues.” If a channel is in MPHY UTOPIA mode, the minimum size should be equal to $\sum^{nMPHY} [\langle NCITSx \rangle + 2]$
0x4C	TQAPTR	Hword	Transmit queue APC pointer. Points to the next available entry in the transmit queue. The APC uses TQAPTR for the channel currently scheduled to transmit. TQAPTR automatically wraps back to TQBASE when it reaches the end of the queue. See Section 39.6, “PHY Transmit Queues.” Initialize TQAPTR to the value in TQBASE.

Table 37-1. Serial ATM and UTOPIA Interface Parameter RAM Map (continued)

<table>
<tr><th>Offset [1]</th><th>Name</th><th>Width</th><th>Description</th></tr>
<tr><td>0x4E</td><td>TQTPTR</td><td>Hword</td><td>Transmit queue transmitter pointer. Points to the next channel to be sent. TQTPTR lags behind the APC pointer (TQAPTR). TQTPTR automatically wraps back to TQBASE when it reaches the end of the queue. See Section 39.6, “PHY Transmit Queues.”
Initialize TQTPTR to the value in TQBASE.</td></tr>
<tr><td>0x50</td><td>APCST</td><td>Hword</td><td>APC state. See Section 37.6, “APC State Register (APCST).”</td></tr>
<tr><td>0x52</td><td>APCPTR</td><td>Hword</td><td>APC priority level base pointer. Points to the base address of the APC priority levels (see Section 39.7, “APC Priority Levels”). APCPTR is an offset from the beginning of dual-port RAM. It should be divisible by 32 (end with 0b00000).</td></tr>
<tr><td>0x54</td><td>AM1</td><td>Hword</td><td rowspan="5">Address match parameters 1–5. The ATM controller provides three methods for address matching: using a lookup table, address compression, or content-addressable memory (CAM). See Section 37.5, “Address Match Parameters (AM1–AM5),” for more information about the configuration of these fields.</td></tr>
<tr><td>0x56</td><td>AM2</td><td>Hword</td></tr>
<tr><td>0x58</td><td>AM3</td><td>Hword</td></tr>
<tr><td>0x5A</td><td>AM4</td><td>Hword</td></tr>
<tr><td>0x5C</td><td>AM5</td><td>Hword</td></tr>
<tr><td>0x5E</td><td>ECSIZE</td><td>Hword</td><td>Expanded cell size/unassigned cell data. Specifies the size of the expanded cells. ECSIZE is valid only if SxSTATE[EC] is set. The expanded header can be from 0 to 12 octets and can optionally include the HEC as shown in the following table:
<table>
<tr><th colspan="3">ECSIZE bits (binary)</th><th colspan="4">Number of Octets</th></tr>
<tr><th>11 (HECI)</th><th>12:13 (bytes)</th><th>14:15 (words)</th><th>Expanded Cell Header</th><th>ATM Cell Header</th><th>HEC/UDF</th><th>Payload</th></tr>
<tr><td rowspan="13">0</td><td>00</td><td rowspan="4">00</td><td>0</td><td rowspan="13">4</td><td rowspan="13">0</td><td rowspan="13">48</td></tr>
<tr><td>01</td><td>1</td></tr>
<tr><td>10</td><td>2</td></tr>
<tr><td>11</td><td>3</td></tr>
<tr><td>00</td><td rowspan="4">01</td><td>4</td></tr>
<tr><td>01</td><td>5</td></tr>
<tr><td>10</td><td>6</td></tr>
<tr><td>11</td><td>7</td></tr>
<tr><td>00</td><td rowspan="4">10</td><td>8</td></tr>
<tr><td>01</td><td>9</td></tr>
<tr><td>10</td><td>10</td></tr>
<tr><td>11</td><td>11</td></tr>
<tr><td>00</td><td>11</td><td>12</td></tr>
<tr><td>1</td><td colspan="2">any combination</td><td>as above</td><td></td><td>1</td><td></td></tr>
</table></td></tr>
<tr><td>0x60</td><td>—</td><td>Word</td><td>Reserved.</td></tr>
</table>

Table 37-1. Serial ATM and UTOPIA Interface Parameter RAM Map (continued)

Offset [1]	Name	Width	Description
0x64	R_PTR	Word	Receiver internal data pointer. Points to the next data location in the receive buffer during cell reception. For internal use.
0x68	RTEMP[2]	Word	Receiver temporary data storage.
0x6C	T_PTR	Word	Transmitter internal data pointer. Points to the next data location in the transmit buffer during cell transmission. For internal use.
0x70	TTEMP[2]	Word	Transmitter temporary data storage.
0x74 to 0x7F	—	12 Bytes	Reserved

Notes: Parameters shown shaded are used for serial ATM only. Non-shaded parameters are used for both serial ATM and UTOPIA operations.
Parameters shown in boldface type must be initialized by the user before enabling ATM operations. Parameters not specified as user-initialized are configured by the CP and should not be modified by the user.

[1] From SCC base. SCC base = IMMR + 0x3C00 (SCC1) or 0x3F00 (UTOPIA)
[2] During transfers, the CP uses SRCHNUM, STCHNUM, OLDLEN, RTEMP, TTEMP, RSTUFF, RHECTEMP, THECTEMP, RSCRAM, RSCRAM1, TSCRAM, TSCRAM1, RCHAN, TCHAN, RCRC and TCRC to store temporary data for internal use.

Table 37-2 describes additional parameters needed to configure the SCC for serial ATM operation.

Table 37-2. Serial ATM Parameter RAM Map

Offset [1]	Name	Width	Description
0xC0	ALPHA	Hword	Receiver delineation alpha/delta counters. The ATM controller applies the HEC delineation mechanism described in ITU specification I.432 where ALPHA and DELTA are initialized by the user to a value from 0 to 63. (The ITU specification I.432 recommendation is 0x7 for alpha and 0x6 for delta.) The receiver updates ALPHA and DELTA; the user should not write to these locations during receive operations.
0xC2	DELTA	Hword	
0xC4	RSTUFF [2]	Word	Receive data stuffing location (for 53 to 52 byte conversion).
0xC8	SHUFFLESTATE	Hword	Receiver data shuffling internal state. Should be cleared during initialization.
0xCA	RHECTEMP[2]	Hword	Receiver temporary HEC storage area
0xCC	THECTEMP[2]	Hword	Transmitter temporary HEC storage area
0xCE	ASTATUS	Hword	Cell synchronization status register. See Section 37.7, “Serial Cell Synchronization Status Register (ASTATUS).” Should be cleared during initialization.
0xD0	HEC_ERR	Hword	HEC error counter. Contains a 16-bit counter for incoming cells with HEC errors. HEC_ERR may be read by the user at any time. Should be cleared during initialization.
0xD2	—	Hword	Reserved
0xD4	RSCRAM[2]	Word	Receiver scrambling storage. Should be cleared during initialization.
0xD8	RSCRAM1[2]	Word	

Table 37-2. Serial ATM Parameter RAM Map (continued)

Offset [1]	Name	Width	Description
0xDC	TSCRAM[2]	Word	Transmitter scrambling storage. Should be cleared during initialization.
0xE0	TSCRAM1[2]	Word	
0xE4	RCRC[2]	Word	Receiver temporary CRC
0xE8	TCRC[2]	Word	Transmitter temporary CRC
0xEC	RCHAN[2]	Word	Receiver current channel
0xF0	TCHAN[2]	Word	Transmitter current channel
0xF4 to 0xFF	—	12 Bytes	Reserved

Note: Parameters in boldface type are initialized by the user before ATM operations. Parameters not specified as user-initialized are configured by the CP and should not be modified by the user.

[1] From SCC base. SCC base = IMMR + 0x3C00 (SCC1)
[2] During transfers the CP uses SRCHNUM, STCHNUM, OLDLEN, RTEMP, TTEMP, RSTUFF, RHECTEMP, THECTEMP, RSCRAM, RSCRAM1, TSCRAM, TSCRAM1, RCHAN, TCHAN, RCRC and TCRC to store temporary data for internal use.

37.1 SAR Receive Function Code Register (SRFCR)

The SAR receive function code register (SRFCR), shown in Figure 37-1, contains the user-initialized function codes and byte ordering information for DMA transfers.

Bit	0		2	3	4	5		7
FIELD	—			BO		FC		
RESET	—	—	—	—	—	—	—	—
OPER	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Figure 37-1. SAR Receive Function Code Register (SRFCR)

The SRFCR fields are described in Table 37-3.

Table 37-3. SRFCR Field Descriptions

Bits	Name	Description
0–2	—	Reserved
3–4	BO	Byte ordering. Program BO to select the required byte ordering for the SDMA transfers. 00 Reserved 01 Modified little-endian. 1x Big-endian or true little-endian.
5–7	FC	Function code. Contains a user-defined value driven on the address type signals AT[1–3] when the SDMA channel accesses memory.

37.2 SAR Receive State Register (SRSTATE)

The SAR receive state register (SRSTATE), shown in Figure 37-2, contains user-initialized global state parameters.

Bit	0	1	2	3	4	5	6	7
FIELD	EXT	ACP	EC	SNC	—	DIS	SER	MPY
RESET	—	—	—	—	—	—	—	—
OPER	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Figure 37-2. SAR Receive State Register (SRSTATE)

The SRSTATE fields are described in Table 37-4.

Table 37-4. SRSTATE Field Descriptions

Bits	Name	Description
0	EXT	Extended channel mode. EXT and ACP select the address matching mechanism; see Section 37.5, “Address Match Parameters (AM1–AM5).” 0 Maximum of 31.5 channels available. (Receive channel 0 is reserved for the raw cell queue.) Channel mapping and connection tables are supported internally. Internal look-up table channel mapping mechanism is used. 1 Maximum of 65534.5 channels available. (Channel 65535 and receive channel 0 are reserved.) Channel mapping and connection tables are supported externally. CAM or address compression is used for channel mapping.
1	ACP	Address compression. Valid only if EXT = 1. EXT and ACP select the address matching mechanism; see Section 37.5, “Address Match Parameters (AM1–AM5).” 0 CAM is used for channel mapping. 1 Address compression mechanism is used for channel mapping.
2	EC	Expanded cell. This option is valid only in UTOPIA mode. 0 Standard 53-byte ATM cell is used. 1 Expanded cells are used on all ATM cells. Cell length can be 52, 56, 60 or 6452 to 65 bytes, as defined by ECSIZE in the parameter RAM.
3	SNC	Synchronization status. Valid only in UTOPIA mode (PDPAR[UT] =1). 0 SOC sync is lost. Also reported via an interrupt in the UTOPIA event register (IDSR1[SYNC]). 1 SOC is in sync. Should be cleared during initialization.
4	—	Reserved
5	DIS	Disable. Used to disable all receive processes in UTOPIA mode. This function is used only when initializing the UTOPIA port. It must not be written to while the UTOPIA port is operating. 0 Receive enabled. 1 Receive disabled. Reserved
6	SER	ATM physical interface type 0 UTOPIA PHY 1 Serial PHY
7	MPY	Enable multi-PHY mode. Valid only for the page 4 parameter RAM when in UTOPIA mode. 0 Single PHY mode 1 Multiple PHY mode

37.3 SAR Transmit Function Code Register (STFCR)

The SAR transmit function code register (STFCR), shown in Figure 37-3, contains the user-initialized function codes and byte ordering information for DMA transfers.

Bit	0	1	2	3	4	5	6	7
FIELD	—			BO		FC		
RESET	—	—	—	—	—	—	—	—
OPER	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Figure 37-3. SAR Transmit Function Code Register (STFCR)

The STFCR fields are described in Table 37-5.

Table 37-5. STFCR Field Descriptions

Bits	Name	Description
0–2	—	Reserved
3–4	BO	Byte ordering. Program BO to select the required byte ordering for the SDMA transfers. 00 Reserved 01 Modified little-endian. 1x Big-endian or true little-endian.
5–7	FC	Function code. Contains a user-defined value driven on the address type signals AT[1–3] when the SDMA channel accesses memory.

37.4 SAR Transmit State Register (STSTATE)

The SAR transmit state register (STSATE), shown in Figure 37-4, contains user-initialized global state parameters.

Bit	0	1	2	3	4	5	6	7
FIELD	EXT	TQF	EC	PBF	—	—	SER	MPY
RESET	—	—	—	—	—	—	—	—
OPER	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Figure 37-4. SAR Transmit State Register (STSTATE)

The STSTATE fields are described in Table 37-6.

Table 37-6. STSTATE Field Descriptions

Bits	Name	Description
0	EXT	Extended channel mode. 0 Maximum of 31.5 channels available. (Receive channel 0 is reserved for the raw cell queue.) Connection tables are supported internally. 1 Maximum of 65534.5 channels available. (Channel 65535 and receive channel 0 are reserved.) Connection tables are supported externally.
1	TQF	Transmit queue full. This bit is set when the APC bypass command is used to indicate that the transmit queue is full. The user must ensure that this bit is not set before issuing an APC bypass command. The transmitter will clear this bit as soon as there is space in the transmit queue. This bit should be initialized to zero (0).
2	EC	Expanded cell. This option is valid only in UTOPIA mode. 0 Standard 53-byte ATM cell is used. 1 Expanded cells are used on all ATM cells. Cell length can be 52, 56, 60 or 64 bytes, as defined by ECSIZE in the parameter RAM.
3	PBF	Port B flag. May be used in multi-PHY mode to coordinate CP and HOST write cycles to their Port B general-purpose signals (if the host is using different port B general-purpose signals). The host uses PBF to notify the CP that PORTB general-purpose signals have been modified. In multi-PHY mode, the CP checks PBF before performing read-modify-write cycles to the Port B multi PHY address signals. If the PBF bit is set, the CP waits for the host to clear the bit. The CP will then start the read-modify-write cycle. The host should set PBF before performing read-modify-write cycles to prevent overwriting the CP configuration of the multi-PHY signals. 0 No-flag. The CP is allowed to perform the read-modify-write cycle. 1 Flag is set. The CP should wait until PBF is clear and then can perform the multi-PHY read-modify-write cycle from port B. The host should set PBF before its port B write cycle and ensure that PBF is cleared for at least 10 system clocks following a host read-modify-write cycle.
4	—	Reserved
5	—	Reserved
6	SER	ATM physical interface type 0 UTOPIA PHY 1 Serial PHY.
7	MPY	Enable multi-PHY mode. Valid only for page 4 of the parameter RAM when in UTOPIA mode. 0 Single PHY mode 1 Multiple PHY mode

37.5 Address Match Parameters (AM1–AM5)

The ATM controller uses one of the three following methods for address matching.

- An internal look-up mechanism—See Table 37-7 for the AM1–AM5 configuration for operation with an address look-up table.
- Address compression—See Table 37-9 for the AM1–AM5 configuration for operation with address compression.
- Content-addressable memory (CAM)—See Table 37-11 for the AM1–AM5 configuration for CAM operation.

The address match parameter configuration when using the internal address look-up table

(EXT = 0) is shown in Table 37-7. See also Section 38.1.1, “Internal Look-up Mechanism (SRSTATE[EXT] = 0).”

Table 37-7. AM1–AM5 Parameters for the Internal Look-up Table

Field	Name	Function
AM1 AM2	HMASK	Header mask. The ATM controller masks the header of each incoming cell with HMASK and uses the resulting masked header in the address match process. The masking process uses a bitwise AND function so bits are masked out by clearing the relevant bits in HMASK. The HMASK fields are shown in Figure 37-5..
AM3	AMBASE	Address matching table base. Used as a pointer to the address lookup table. It is scanned from the top (AMEND) to the base, so headers for the busiest connections should be at the top of the table. When a match occurs, the CP uses the location of the match to locate the channel number in the pointer table. Initialize AMBASE to point to the last entry of the lookup table. AMBASE must be word-aligned.
AM4	AMEND	Address matching end pointer. Contains the address of the top entry in the lookup table. New connections are added to the table at the location immediately above AMEND. The match process starts at the top (AMEND) and proceeds to the base (AMBASE). AMEND is maintained by the user.
AM5	APBASE	Address pointing table base. Contains the base address of a table of pointers to the connection tables. Initialize APBASE to point to the last pointer of the pointing table, which contains the offset of the connection table for the global raw cell queue (channel 0). APBASE must be halfword-aligned.

HMASK is shown in Figure 37-5.

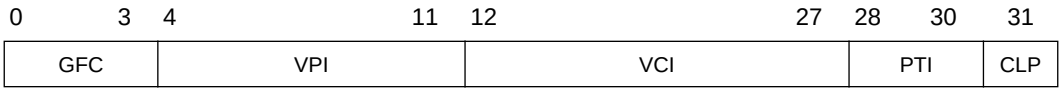


Figure 37-5. HMASK Cell Header Mask Fields

The HMASK fields are described in Table 37-8.

Table 37-8. HMASK Field Descriptions

Bit(s)	Name	Description
0–3	GFC	Generic flow control. Can be cleared to indicate that GFC protocol is not enforced.
4–11	VPI	Virtual path identifier mask
12–27	VCI	Virtual channel identifier mask
28–30	PTI	Payload type identifier mask
31	CLP	Cell loss priority mask

The address match parameter configuration for extended channel mode with address compression (EXT = 1 and ACP = 1) is shown in Table 37-9. See also Section 38.1.2, “Address Compression (SRSTATE[EXT,ACP] = 11).”

Table 37-9. AM1–AM5 Parameters for Extended Channel Address Compression

Field	Name	Function
AM1	FLBASE	First-level table base. Contains the word-aligned starting address for the first-level table of the address compression mechanism.
AM2		
AM3	SLBASE	Second-level table base. Pointer to the beginning of a 64-Kbyte memory space where the set of second-level addressing tables are located.
AM4		
AM5	FLMASK	First-level mask. The ATM controller masks the GFC, VPI, and PTI bits of the header of each incoming cell with FLMASK[1–15] and uses the resulting masked header in the first-level address matching process. The masking process uses a bitwise AND function to allow address bits to be masked out by clearing the relevant bits in FLMASK. The FLMASK fields are shown in Figure 37-6.. The FLMASK should contain an unbroken sequence of ones. For example, the sequence 0b0000_0011_1111_1110 would be a valid sequence, while the sequence 0b0000_1111_1001_1100 contains a broken sequence of ones and would lead to undefined behavior during the matching process.

FLMASK is shown in Figure 37-6.

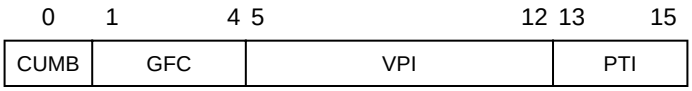


Figure 37-6. FLMASK

The FLMASK fields are described in Table 37-10.

Table 37-10. FLMASK Field Descriptions

Bits	Name	Description
0	CUMB	Check unused mask bits. CUMB allows the user to screen out (pass to the global raw cell queue) misinserted cells. Setting CUMB signals the receiver to check for non-zero values in the address bits not used during the address matching. See Section 38.1.2.4, “Preventing Channel Aliasing.” 0 Do not check unused address bits. 1 Check that all unused address bits equal 0.
1–4	GFC	Generic flow control mask. Can be cleared when the GFC protocol is not enforced.
5–12	VPI	Virtual path identifier mask
13–15	PTI	Payload type identifier mask

The address match parameter configuration for extended channel mode CAM operation (EXT = 1 and ACP = 0) is shown in Table 37-11. See also the discussion in Section 38.1.3, “CAM Address Mapping (SRSTATE[EXT,ACP] = 10).”

Table 37-11. AM1–AM5 Parameters for Extended Channel CAM Operation

Field	Name	Function
AM1	HMASK	Header mask. The ATM controller masks the header of each incoming cell with HMASK and uses the resulting masked header for address matching. The masking process uses a bitwise AND function so bits are masked out by clearing the relevant bits in HMASK. The HMASK fields are shown in Figure 37-5..
AM2		
AM3	CAMADD	CAM address. DMA pointer to the CAM's address in the memory map. The DMA uses the CAM address for read and write cycles. CAMADD should be divisible by 16.
AM4		
AM5	CAMLEN	CAM length. Contains the number of entries in the CAM.

37.6 APC State Register (APCST)

The APC state register (APCST), shown in Figure 37-7, contains state and control parameters for the APC mechanism.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	—	CSER		NSER		NMPHY			CMPHY			—	—	DIS	PL2	MPY
RESET	0	UD		UD		UD			UD			0	0	0	UD	UD
OPER	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	0x50 in SCC parameter RAM															

Figure 37-7. APC State Register (APCST)

The APCST fields are described in Table 37-12.

Table 37-12. APCST Field Descriptions

Bits	Name	Description
0	—	Reserved
1–2	CSER	Current serial ATM or UTOPIA port. CSER is used by the APC to point to the currently active serial or UTOPIA interface if multiple physical ATM ports are active. Initialize CSER with the same value programmed in NSER.
3–4	NSER	Next serial ATM or UTOPIA port. NSER points to the next serial ATM or UTOPIA port to be serviced after the APC completes servicing this ATM port. By programming NSER for each ATM port, the user builds a linked list of port numbers and thereby determines the order in which the ports are serviced. (NSER should point only to SCC operating in ATM mode.) See also Section 39.3, "Using the APC with Multiple ATM Ports," and Section 39.4, "Using the APC Without Using UTOPIA." 00 Serial port 1 (SCC1) 01 Reserved 10 Reserved 11 Page 4 or UTOPIA port (SCC4) Note that if only one ATM port is used (through either page 4 or UTOPIA), program NSER to 0b11 (so that the port points to itself to be serviced next).

Table 37-12. APCST Field Descriptions (continued)

Bits	Name	Description
5–7	NMPHY	Number of (multiple) PHYs. Valid only for the page 4 parameter RAM when in UTOPIA multi-PHY mode; in all other parameter RAM pages (SCC1–3), this field should be cleared. NMPHY specifies the total number of PHY devices connected to the UTOPIA port. 000 1 PHY 001 2 PHYs 010 3 PHYs 011 4 PHYs 1xx Reserved
8–10	CMPHY	Current multi-PHY. Valid only for the page 4 parameter RAM when in UTOPIA multi-PHY mode. CMPHY is used only by the CP and should be initialized with the same value programmed in NMPHY.
11	—	Reserved
12	—	Reserved
13	DIS	APC disabled status flag. Valid for serial ATM only. Set by the transmitter when a global FIFO underrun (GUN) exception occurs. Should be cleared during initialization.
14	PL2	Priority level 2. Enables the second APC priority level. 0 Disable the second-level APC scheduling table. 1 Enable the second-level APC scheduling table.
15	MPY	Enable multi-PHY mode. Valid only for page 4 of the parameter RAM when in UTOPIA mode. For additional information about the multi-PHY configuration, see Section 38.2, “Multi-PHY Configuration (MPHY).” 0 Single PHY mode 1 Multiple PHY mode

37.7 Serial Cell Synchronization Status Register (ASTATUS)

The serial ATM cell synchronization status register (ASTATUS) provides status information concerning FIFO errors and receiver synchronization status. The ASTATUS register is shown in Figure 37-8.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
—	—	—	—	—	—	ORUN	URUN	—	—	—	—	—	—	—	LOCK

Figure 37-8. Serial Cell Synchronization Status Register (ASTATUS)

The ASTATUS fields are described inTable 37-13.

Table 37-13. ASTATUS Register Field Descriptions

Bits	Name	Description
0–5	—	Reserved
6	ORUN	Receiver FIFO overrun. The CP sets this flag to indicate a receiver overrun event has occurred. The user can acknowledge the flag by clearing it. 0 No receiver FIFO overrun 1 Receiver FIFO overrun. Note that the SCCE[GOV] bit is also set.
7	URUN	Transmitter FIFO underrun. The CP sets this flag to indicate a transmitter underrun event has occurred. The user can acknowledge the flag by clearing it. 0 No transmitter FIFO underrun 1 Transmitter FIFO underrun. Note that the SCCE[GUN] bit is also set.
8–14	—	Reserved
15	LOCK	Cell delineation status. Maintained by the CP; should be read-only for the user. Indicates the current cell delineation status. Should be cleared during initialization. 0 The receiver is out of synchronization and is not receiving cells. 1 The receiver has gained cell delineation and is receiving cells. Note that the receiver cell delineation status is also indicated by SCCE[SYNC].

Chapter 38
ATM Controller

This chapter describes the address mapping mechanisms of the ATM controller to support connection tables for both single- and multi-PHY interfaces, and the commands provided to control ATM transmit and receive operations on a channel-by-channel basis.

38.1 Address Mapping

Three methods for mapping incoming cell header addresses to local ATM channel numbers are available. The first method is based on an internal look-up table and is used when the maximum number of ATM receive connections can be handled by the internal channels. When more than 31 receive connections are required (extended channel mode), two methods for address mapping are supported: address compression and the use of content addressable memory (CAM).

Only one address mapping method can be used at a time, and it should be selected in the parameter RAM (see the EXT and ACP field descriptions in Section 37.2, “SAR Receive State Register (SRSTATE)”) during system start-up. The following sections describe each address mapping method.

38.1.1 Internal Look-up Mechanism (SRSTATE[EXT] = 0)

The internal look-up mechanism maps the address fields in the header of incoming cells to internal channel numbers. This mapping mechanism uses two tables: an address matching table and a pointing table. The matching table contains up to thirty-one 32-bit address (GFC/VPI/VCI/PTI/CLP) entries and one empty entry at the base of the table reserved for the raw cell queue. (Generally, channel number 0 is used for the raw cell queue as a convention.) The pointing table contains thirty-two 16-bit RCT base address pointers corresponding one-to-one with the address matching table entries.

When a cell is received, the cell header is masked by performing bitwise AND logic with HMASK in the parameter RAM (see Table 37-8). The resulting masked header is then compared with each entry in the matching table starting from the top (AMEND). When an entry matches, the parallel entry in the pointing table is then read to get the RCT*n* base address of the local channel *n* assigned to the incoming cell’s address. If, however, the

address matching algorithm reaches the bottom of the table (AMBASE) without a match, the incoming cell is directed to the global raw cell queue; that is, the base address of RCT0 should be placed in the corresponding last entry of the pointing table (APBASE).

Note that the internal look-up mechanism searches the address matching table sequentially from the top (AMEND) to the base (AMBASE). Therefore, it is recommended to put the most frequently used connections at the top of the table (near AMEND). Note that AMEND points to the lowest address and that new channels are added above this pointer.

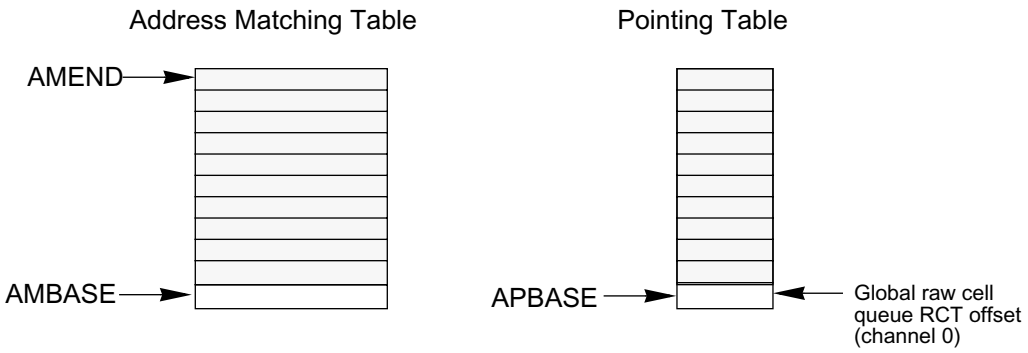


Figure 38-1. Address Mapping Tables for Internal Channels

38.1.1.1 Adding a New Internal Channel

A new internal channel is added to the internal look-up mechanism on-the-fly in three steps:

1. Add an RCT address pointer at the top of the pointing table.
2. Add an address match entry to the top of the matching table (above AMEND).
3. AMEND is updated to point to the new entry. (This last step allows the CP to begin using the new address entry for matching.)

38.1.1.2 Removing an Internal Channel

A channel entry is removed from the internal look-up mechanism on-the-fly in three steps. Note that a channel can be removed only when it has stopped receiving cells (the upstream ATM channel has stopped sending cells). Remove a channel entry as follows:

1. Copy the top entry of the match table (the address match entry specified by the AMEND pointer) to the position to be deleted.
2. Copy the top pointer of the pointing table (that is, the pointer parallel to AMEND) to the position to be deleted.
3. Advance the AMEND pointer one entry toward AMBASE.

38.1.2 Address Compression (SRSTATE[EXT,ACP] = 11)

The address compression mechanism uses two levels of address translation to help minimize the memory space needed to cover the available address range. In the first-level compression, the GFC, VPI, and PTI fields of the received header are masked with FLMASK to create a pointer (offset from FLBASE) to the first-level addressing table. The first-level table (FLT) contains an additional mask and table pointer to one of the second-level tables (SLTs), referred to as the second-level table offset (SLTOFFSET). (SLTOFFSET is an offset from the base address of the second-level tables (SLBASE). In the second-level compression, the VCI bits are masked with the SLMASK field from the first-level table, and the result is used as an index pointer into the particular SLT addressed by SLTOFFSET. The SLT entry contains the assigned local channel number for the received cell.

FLBASE, SLBASE and FLMASK are defined in the parameter RAM; see Table 37-10. The following sections describe the addressing tables and show an example of address compression.

38.1.2.1 First-Level Addressing Table (FLT)

Each entry in the first-level addressing table (FLT) contains a 16-bit second-level mask (SLMASK) which is used to mask the incoming cell's VCI. SLMASK should contain a contiguous sequence of ones that operates in the same way as the FLMASK bit sequence. Note also that SLMASK must contain at least one bit set. Failure to do so results in having the cells associated with the corresponding VP (VPI x FLMASK) routed to the PHY's raw cell queue. The FLT entry also contains a 16-bit second-level table offset (SLTOFFSET) that points to a single SLT.

Unused FLT entries should be cleared (null entry). Cells with a null entry pointer are received into the global raw cell queue.

The size of the FLT depends on the number of mask bits in the FLMASK. If, for example, FLMASK contains an unbroken sequence of ten bits set, the index pointer into the FLT will contain 10 bits, resulting in a table size of 4 Kbytes. The actual address of an FLT entry is FLBASE + (index_pointer x 4).

38.1.2.2 Second-Level Addressing Tables (SLTs)

An SLT entry contains the 16-bit local channel number (0-65534) assigned to match a received cell header's VCI and VPI. The local channel number corresponds to an RCT, where:

- Channel number 0—Reserved for the global raw cell queue. RCT0 is located in the dual-port RAM pointed to by CT_BASE.
- Channel number 1–31—RCTs for these connections are in the dual-port RAM. The address of each RCT is (channel_number x 64 + CT_BASE).

- Channel numbers greater than 31—RCTs for these connections are in external memory. The address of each RCT is (channel_number x 64 + ECT_BASE).

The number of entries in each SLT depends on the length of the sequence of ones in SLMASK. For example, the size of an SLT with a sequence of 10 bits set in the SLMASK is 2 Kbytes. The actual address of an entry in an SLT is equal to (SLBASE + SLOFFSET + index_pointer x 2).

38.1.2.3 Address Compression Example

Figure 38-2 shows an example of address compression. The first-level mask (FLMASK) selects the third PTI bit and five VPI bits. Bitwise ANDing of the FLMASK with the GFC, VPI, and PTI bits results in a 6-bit pointer. Pointer1 turns out to equal 0x3, and therefore is pointing to the fourth entry of the FLT. The entry contains a 16-bit mask (SLMASK) for the VCI field, and the second-level table offset (SLTOFFSET) pointing to one of the SLTs. The VCI is masked with SLMASK, resulting in a 7-bit pointer to the assigned channel number in the SLT.

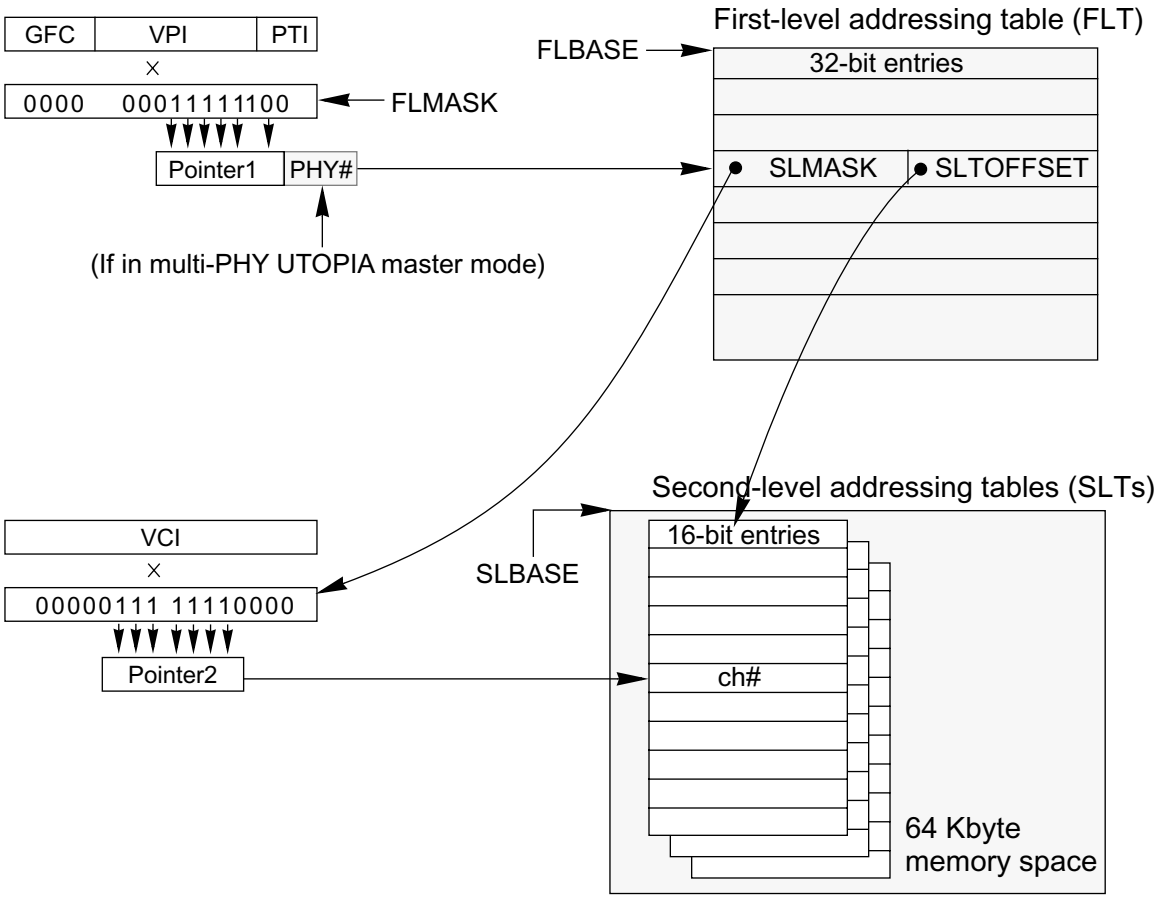


Figure 38-2. Address Compression

38.1.2.4 Preventing Channel Aliasing

Reliable one-to-one mapping of VCs to local channel numbers requires that the address bits not taken into account during the translation have a fixed value (chosen to be zero). Otherwise, multiple VCs could translate to a single local channel number. The CUMB feature (check unused mask bits) can be used to test the reliability of the mapping by screening out misinserted cells. When FLMASK[CUMB] is set, address header bits not used in the first- and second-level address masking procedure are checked for non-zero values. If a non-zero value is found, the cell is passed to the global raw cell queue. See Table 37-11 for a description of the CUMB bit.

Note that if CUMB is set, the user should also include the PTI bits in FLMASK so that cells marked as congested (EFCI = 1) or last (PTI[1] =1) in the PTI are not received into the global raw cell queue.

38.1.2.5 OAM Screening

OAM/management cells are indicated by having their most significant PTI bit set. If OAM/management cells are not a part of the address mapping process (the most significant PTI bit in FLMASK is cleared), OAM cells are passed to the global raw cell queue instead of being routed to the buffer otherwise assigned to this incoming channel. This treatment keeps the OAM cells of an AAL5 connection from interfering with the SAR process and allows OAM cells to be processed separately.

38.1.3 CAM Address Mapping (SRSTATE[EXT,ACP] = 10)

The CAM address mapping method uses dual DMA accesses to an external CAM-based look-up table. When a cell is received, the incoming header is masked with HMASK in the parameter RAM (see Table 37-12). The CP then performs a DMA write access to the CAM address (CAMADD) with the address generated from the masked header as its data operand. The second access is a read DMA access to CAMADD. During the read access, the CAM should drive a “match successful” indication on the data bus D[0] signal, and the matched channel number on the data bus D[16–31] signals. The “match successful” indication driven on data bus D[0] is active low, (a 0 indicates a successful match). If there is no “match successful” signal driven, the cell is passed to the global raw cell queue.

38.2 Multi-PHY Configuration (MPHY)

The MPC855T can handle up to 4 different PHYs in UTOPIA mode. The interface to the PHYs is done through the UTOPIA and 4 additional port B signals. The configuration of parameters for multi-PHY operations is described in this section.

38.2.1 Setting Multi-PHY mode

To initiate multi-PHY mode the user must set the MPY bit in the SRSTATE, STSTATE and APCST fields in the parameter RAM. The user must also select UTOPIA mode (through the PDPAR[UT] bit) and set the number of PHYs in the NMPHY field of APCST.

When operating in multi-PHY mode the two least-significant bits of the channel number entry in the pointing/address tables represent the associated PHY number. The format of the Pointing Table, Second Level Addressing Table, or CAM entries is shown in Figure 38-3. This is also the format used when activating a multi-PHY channel. The transmit queue entries have this format (needed to identify the transmitting channel's PHY).

Figure 38-3. Multi-PHY Pointing Table Entry

38.2.2 Receive Multi-PHY Operation

A receive operation starts with the PHY address driven on PHREQ[0–1] and the assertion of RxCav by one of the PHYs. The MPC855T reads the PHY number through PHREQ[0–1] and writes the selected address to PHSEL[0–1] before to the assertion of $\overline{\text{RxEnb}}$.

38.2.2.1 Look-up Table MPHY Support

When performing multi-PHY operations the user must prepare up to 4 separate look-up and pointing tables (thereby providing a table set for each PHY). The AMBASE, APBASE and AMEND values used for all PHYs during multi-PHY operations are configured as shown in the following calculations:

$$AMBASE\{N\} = AMBASE + N \times 8 \times 4$$

$$AMEND\{N\} = AMEND + N \times 8 \times 4$$

$$APBASE\{N\} = APBASE + N \times 8 \times 2$$

In the preceding calculations, N represents the PHY number. When performing multi-PHY operations using the internal dual-port RAM, each PHY may use up to 8 channels (as only 32 channels are available in the internal dual-Port RAM). Note that one channel out of each group of 8 must be reserved for the global raw cell queue. If the extended channel mode operation is selected, more than 8 channels per PHY are supported. Figure 38-4 shows the configuration of the address mapping tables for multi-PHY operations.

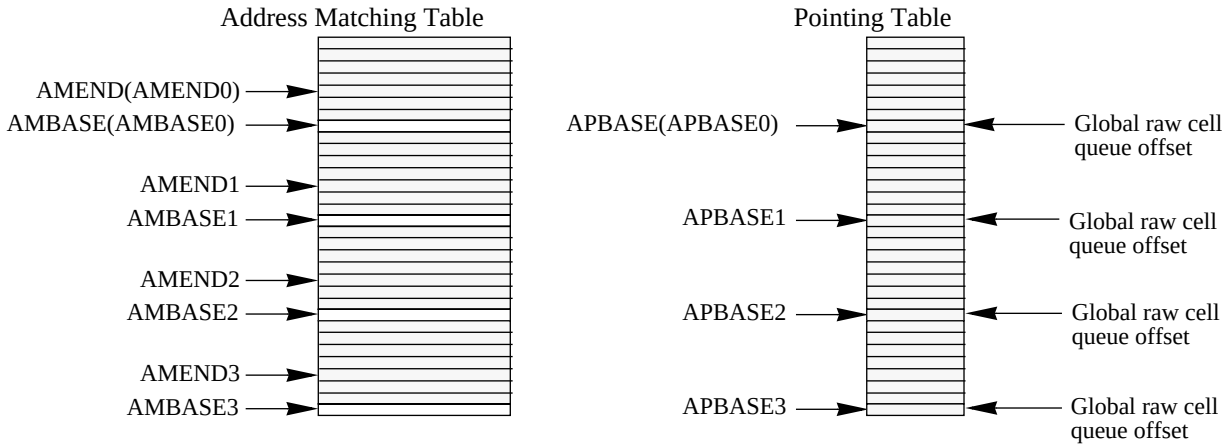


Figure 38-4. Address Mapping Tables for Multi-PHY Operations

Note that the address in the AMEND field is common to the four look-up tables. AMEND points to the highest valid channel number in any one of the four tables. For example, if PHY3 handles 5 channels and the other PHYs handle only 2 channels, the address in AMEND should be set to point to the fifth channel of PHY3. The unused address pointers in PHY0, PHY1, and PHY2 should point to the raw cell queue.

38.2.2.2 Address Compression Multi-PHY Support

During multi-PHY operations the 2-bit PHY address is appended as the least significant bits of the first-level address pointer.

38.2.2.3 CAM Multi-PHY Support

When performing CAM addressing the PHY address is added to the CAMADD address. The user must configure the CAMADD field with the last 5 address bits cleared. The PHY address is driven on the ADDR[28–31] signals during the CAM access. This allows the user to use either 4 separate CAMs with each mapped to its own address, or to have a single unified CAM with the ADDR[28–31] signals used as part of the match data for the CAM.

38.2.3 Transmit Multi-PHY Operation

A transmit operation starts with the assertion of the TxCav signal. This signal signals the ability of all PHY interfaces to load a cell in their transmit FIFOs. The MPC855T indicates the PHY selected for transmission by writing the PHY number on PHSEL[0–1] prior to assertion of $\overline{\text{TxEnb}}$.

38.2.4 APC Multi-PHY Parameters

Each PHY in multi-PHY mode has a dedicated APC. The APC parameter table is extended to include the parameters for each APC. The APCPTR field in the parameter RAM points to the first table (MPHY0), and the parameter tables for the remaining PHYs are attached to the bottom of the first table. For example, the address (APCPTR+32) will point to MPHY1 parameter table, (APCPTR+64) will point to MPHY2 parameter table, and (APCPTR+96) will point to MPHY3 parameter table.

38.3 ATM Commands

The host application issues commands to the ATM controller by writing to the CP command register (CPCR). The ATM commands are similar to the CP commands provided for other protocols (see Section 18.5.3, “CP Command Register (CPCR)”). A unique CP opcode is assigned for all ATM commands, and a separate ATM opcode selects the specific ATM command. The CPCR format for ATM commands is shown in Figure 38-5.

Note that the worst case ATM command execution latency is 480 clocks, and the typical command execution latency is 40 clocks for serial ATM and 180 clocks for UTOPIA.

BITS	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	RST	ATM OPCODE			OPCODE = 1111[1]				CH_NUM[2]					APCLEV		FLG
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	9C0															

Notes:

1. Opcode for all ATM commands = 1111.
2. The ATM commands have a channel (communications controller) number associated with them. The specific ATM channel number should be written by the user in the COMM_CH field in the controller’s parameter RAM before writing the command into the CPCR.

Figure 38-5. CP Command Register (CPCR) (ATM-Specific)

Table 38-1 describes the configuration of the CPCR for ATM operations.

Table 38-1. CPCR ATM-Specific Field Descriptions

Bits	Name	Description
0	RST	CP reset command. Set by the core and cleared by the CP. Executing this command clears RST and FLG within two general system clocks. The CPM reset routine takes approximately 60 clocks, but CPM initialization can start immediately after this command is issued. Use RST to reset the registers and parameters for all the controllers, as well as the CPM and RISC timer table. RST does not, however, affect the serial interface or parallel I/O registers. 0 No reset issued. 1 Reset issued.
1–3	ATM OPCODE	ATM opcode. Contains the 3-bit opcode of the channel command. See Table 38-2. 000 Transmit channel activate 001 Transmit channel deactivate 010 Stop transmit (abort) 011 Restart transmit 100 Stop receive 101 Restart receive 110 APC bypass
4–7	OPCODE	Opcode. Should contain 0b1111 for ATM operation.
8–11	CH_NUM	Channel (communications controller) number. Specifies the SCC for a command issued to a serial ATM channel. For a UTOPIA channel, CH_NUM should specify UTOPIA. 0000 SCC1 0100 Reserved 1000 Reserved 1100 UTOPIA All others are reserved.
12	—	Reserved
13–14	APCLEV	APC Level. This field is valid only for the transmit activate command. APCLEV specifies which level of the APC table the current channel should be inserted. 00 Insert the channel to first-level APC table. 01 Insert the channel to second-level APC table. 1x Reserved
15	FLG	Command semaphore flag. Set by the core and cleared by the CP. 0 CP is ready for a new command. 1 CP is currently processing a command—cleared when the command is done or after reset.

The ATM commands are described in Table 38-2.

Table 38-2. ATM Commands

Command	ATM Opcode	Description
transmit activate channel	000	Activates the channel specified in COMM_CH by inserting its channel number into the APC scheduling table at the location indicated by the service pointer. The channel most recently inserted is the first to be chosen by the APC. This command can be issued only after initializing the channel by preparing valid BDs and a TCT.
transmit deactivate channel	001	Deactivates the channel specified in COMM_CH by extracting its assigned channel number from its APC scheduling table. This command should not be issued before the CP completes sending all the BDs associated with the current channel number. The channel number's TCT can be assigned to a different ATM channel only after this command has taken effect.
STOP TRANSMIT (Abort)	010	Instructs the transmitter to stop the channel specified in COMM_CH. Channels are stopped on cell boundaries. Transmission stops after the channel's current cell is next scheduled by the APC. After the current cell is sent, the BD is closed, a new entry to the interrupt queue is optionally added for the channel, the BD pointer is advanced to the next BD, and the channel-disabled flag TCT[CDIS] is set. For AAL5 channels, instead of the current cell, an abort cell (a last cell with PTI[0] = 1 and zero length) is sent to terminate the current frame. After the STOP TRANSMIT command is completed, the stopped channel is still scheduled by the APC, but the channel number is not forwarded to the transmit queue, effectively leaving a hole in the transmit cell stream. Issue a RESTART TRANSMIT command to restart the channel.
restart transmit	011	Restarts transmissions following a STOP TRANSMIT command. This command is used to restart the channel number specified in COMM_CH at the current BD.
stop receive	100	Stops the receiver from receiving cells for the channel specified in COMM_CH. The channel-disabled flag RCT[CDIS] is set, and the in-frame flag RCT[INF] is cleared. The current buffer remains empty; that is, RBD_PTR (or PTP_BD_PTR for PTP connections) is not advanced to the next BD. Cells received for this channel number after the STOP RECEIVE command executes are discarded.
restart receive	101	Restarts the receiver for the channel specified in COMM_CH. RCT[CDIS] is cleared, and the receiver begins receiving cells into the data buffer pointed to by RBD_PTR (or PTP_BD_PTR for PTP connections).
apc bypass	110	Inserts the channel number specified in COMM_CH directly into the transmit queue. Enables out-of-rate cell transmission with cell pacing determined by the user. Make sure that the combined bit rate of all transmitted channels (APC-scheduled and out-of-rate channels) does not exceed the maximum allowed by the PHY. The channel number inserted by the APC_BYPASS command is placed at the front of the transmit queue and therefore becomes the next channel sent. Before activating this command, make sure the TQF bit in the STFCR is not set. The transmit queue should contain at least 2 entries when this command is issued.

Chapter 39
ATM Pace Control

The ATM layer performs cell multiplexing and demultiplexing. The ATM pace control unit (APC) is part of the ATM cell multiplexing process. The APC processes the traffic parameters of each channel and defines the multiplex timing for all the channels. Cell multiplexing is done by the transmitter according to the traffic control function implemented by the APC.

39.1 APC Algorithm

The APC consists of the following major parts:

- APC timer (CPM timer 4), supplying a global tick to the APC process
- APC scheduling tables, data structures used by the APC algorithm for scheduling channels at programmed intervals
- APC pace parameters (APC_Pace), defined in the TCT for each channel
- Transmit queue to hold the scheduled channels (output of the APC algorithm)

The APC is a CPM process that dynamically reads the traffic shaping parameters of each active transmit channel from the TCTs and uses a periodic table-scanning algorithm to determine the next channels to be scheduled. Having identified the next channels, it then places the channel numbers of the selected channels into the transmit queue. The order in which the APC places channel numbers in the transmit queue is the order in which cells are multiplexed onto the line.

The transmit process is separate from the APC algorithm, which exchanges information with the APC via the transmit queue. When the transmit process receives a request from the physical layer (either the SCC or the UTOPIA interface), the transmitter reads the next channel number scheduled in the transmit queue. The transmitter sends one cell for every channel number appearing in the transmit queue. If the transmit queue is empty, the transmitter does nothing for UTOPIA or sends idle cells for serial ATM.

Figure 39-1 shows the APC process and transmit flow in UTOPIA mode. The APC obtains a channel’s pacing information from its TCT so that it can be re-scheduled, and writes the channel number to the transmit queue. For AAL5 channels, the transmitter implements AAL and SAR functions on the external memory for the chosen channel. The transmitter

then sends the cell through the UTOPIA interface to the PHY. The PHY implements the transmission convergence (TC) layer and the physical media-dependent (PMD) layer.

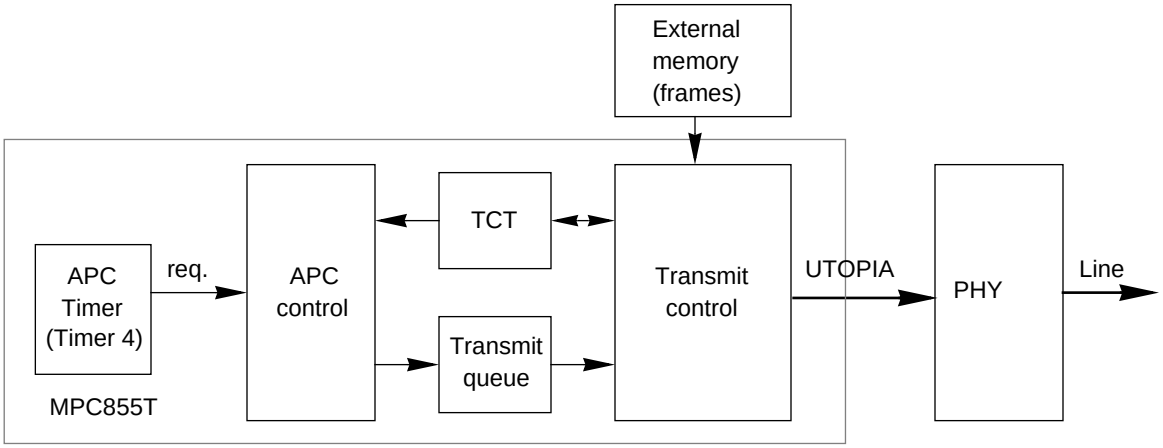


Figure 39-1. APC in UTOPIA Mode—Transmit Flow

39.1.1 APC Implementation

The APC is based on a dynamic scheduling table that contains pointers (implemented as channel numbers) to the active channels. The rate at which the table scan encounters a channel number in the scheduling table determines the transmit rate for that particular channel.

Each APC priority level's scheduling table consists of an array of half-word channel numbers, and two scan pointers (APCT_PTRx and APCT_SPTRx). Each table entry represents a time slot in which the number of cells specified by the parameter NCITS are sent. During initialization, each entry of the APC scheduling table should be programmed with 0xFFFF, which is reserved as the invalid channel number. Issuing the TRANSMIT ACTIVATE CHANNEL command to the CPM causes a channel's number to be inserted into the APC scheduling table at the entry pointed to by the table scan pointer (APCT_PTRx).

Periodic timeouts of the APC timer activate the APC algorithm and cause the real-time scheduling table scan pointer (APCT_PTRx) to advance one entry. Each time APCT_PTRx steps to the next entry in the table, the APC reads each new channel number and schedules it again for a future time slot, according to the APC pace parameter in each channel's TCT.

The entries of an APC scheduling table are actually the heads of linked lists. That is, if more than one channel is scheduled to the same time slot, the first channel points to the next channel using the APC link field (APCL) in its TCT, and so on.

Having scheduled the channels at the current table entry (APCT_PTRx), the APC then inserts up to NCITS channel numbers into the transmit queue using the service pointer (APCT_SPTRx). If more than NCITS channel numbers are scheduled for the same time slot, the leftover channel numbers remain pending until the next time the APC is activated.

The lagging APCT_SPTRx service pointer keeps track of the pending channels so that cells are only deferred, not dropped.

If the highest priority APC level cannot provide NCITS cells, the APC begins traversing the lower priority APC levels hunting for additional channels (advancing each APC level's service pointer in turn) until a total of NCITS cells are found or until no APC levels remain. However, because traversing the APC levels could potentially cause too much delay, the APC_MI (maximum iteration) parameter is provided to limit the total number of times the APC advances a service pointer.

If a channel scheduled for transmission has no buffer descriptors ready, nothing is inserted into the transmit queue. This results in one or more idle cells in the cell stream.

39.1.2 APC Parameters

The APC mechanism can be programmed to provide a wide variety of transmit rates and support a large number of channels. There are several important parameters which define its capabilities. These include:

- Cell scheduling rate. This is determined by the timeout rate of the APC timer (CPM timer 4), which defines the period of the scheduling time slots, and the number of cells transmitted in a time slot (NCITS). The maximum of this rate is the bit rate of the physical medium. However, it could also be any amount less than that, if the user desires to use only a certain percentage of the bandwidth.
- Maximum and minimum bit rates supported for particular channels. A channel number can appear only once in a time slot, but must appear at least once during each table scan. The maximum bit rate for a particular channel is achieved when it is scheduled for transmission in every APC scheduling table entry (i.e. APC Pace = 1). The minimum bit rate for a particular channel is achieved when it is scheduled for transmission only once in a table scan (i.e. APC Pace = APC_table_size - 1). These constraints define the upper and lower bounds at which the transmit rate of a particular channel can be scheduled.
- APC scheduling table size and CPM performance. If the NCITS parameter is increased, the APC scheduling table size will decrease (thereby conserving dual-port RAM space) and the APC timer will make fewer requests to the CPM (thereby decreasing CPM processing overhead). However, increasing NCITS decreases the maximum bit rate supported per channel and increases cell delay variation.

Trade-off decisions must be made when programming these parameters. The following subsections provide examples of the analysis required to make these determinations.

39.1.3 Programming APC Scheduling Table Size and NCITS

The size of the APC scheduling table is defined by the minimum bit rate desired for a single connection and the number of cells transmitted in a time slot.

Defining:

P = cell scheduling bit rate (usually equal to the PHY bit rate)

min_rate = minimum data rate at which a channel can be scheduled

max_rate = maximum data rate at which a channel can be scheduled

M = minimum APC scheduling table size allowable for a particular configuration

The maximum transmit data rate for a single channel supported by the APC is:

$$\text{max\_rate} = \frac{P}{NCITS}$$

The minimum transmit data rate for a single channel supported by the APC is:

$$\text{min\_rate} = \frac{P}{(M-1)NCITS}$$

For example, assuming a PHY transceiver with a transmit data rate of 51.84 Mbps, and assuming that the APC is to be configured to use all of this transmit bandwidth, then P = 51.84Mbps. Also assume that no single ATM channel (virtual connection) will ever require more than 25% of this bandwidth. By this assumption, (P / max_rate) is 4, and therefore NCITS is chosen to be 4. Also, because the application has no single connection requiring a bandwidth less than 32kbps (i.e. min_rate = 32kbps), the minimum APC scheduling table size is therefore equal to:

$$M = \frac{\frac{51.84Mbps}{32kbps}}{4} + 1 = 406$$

This example deals entirely in integers; however, note that while M must be an integer, NCITS may include an integer and a fraction. In any other application of this analysis, NCITS may be tuned in order to achieve an exact integer value M.

Note that a fractional value of NCITS does not mean that a fractional number of channel numbers will be written to the transmit queue. It means that the number of cells scheduled per APC timeout will vary around the average defined by NCITS. For example, if NCITS were programmed to 2.5, then 2 cells would be scheduled during one iteration of the APC algorithm, followed by 3 cells the next time, followed by 2, and so on.

39.1.4 Defining APC Slot Time

The APC defines the maximum bit rate of the cell scheduler through the period of the APC timer tick and the number of cells scheduled per APC timer tick (NCITS). The period of the APC timer is referred to as an APC time slot.

Defining:

P = cell scheduling bit rate (usually equal to the PHY bit rate)

APC_timer_per = CPM timer 4 period, programmed in TRR4 and TMR4

Then:

$$P = \frac{CLKOUT}{(APC\_timer\_per)}(NCITS)(number\_of\_bits\_per\_cell)$$

Using the previous example, if CLKOUT=50MHz, P=51.84Mbps, and NCITS=4, then APC_timer_per is approximately 1635.8. The closest value available using TMR4 only (TRR4 is programmed to 0x1) is 103*16 = 1648, by programming TMR4[PS] = 0x67 and TMR4[ICLK]=0b10. Be sure to choose the next largest period value from that which was calculated; otherwise, the transmit queue would eventually overrun as the APC scheduler would provide slightly more traffic than the physical layer can transmit.

39.1.5 Programming Rates for Channels

The bit rate for a particular ATM channel is defined by that channel's APC pace parameter in its TCT. The APC pace value (APCP + (APCPF/65536)) for any channel must fall between 1 and (APC scheduling table size -1). Values outside this range result in erratic pace and/or scheduling table overflows (APCO interrupts). The maximum rate for transmission of a particular channel is achieved when the APC pace is equal to one.

Defining:

P = cell scheduling bit rate (usually equal to the PHY bit rate)

des_rate = desired bit rate for this channel

Then:

$$APC\_Pace = \frac{P}{(NCIST)(des\_rate)}$$

For example, again using the previous example with P=51.84Mbps and NCIST=4, assume that a channel with a bit rate of 100kbps is desired. APC_Pace should therefore be programmed to 51.84Mbps / (4 * 100kbps) = 129.6. This can be approximated by programming APCP=129 and APCPF=39322.

For another example, assume the desired bit rate is 10Mbps. Then APC_Pace should be programmed to 51.84Mbps / (4 * 10Mbps) = 1.296. This can be approximated by programming APCP=1 and APCPF = 19399.

Note that APC_Pace consists of an integer and a fraction. A channel with a non-integer APC_Pace will be scheduled such that its average pace will be as defined by APCP and APCPF. For example, if APCP and APCPF were programmed to define a pace of 1.5, then

after transmission the channel would be alternately rescheduled either one time slot following or two time slots following, averaging to 1.5.

39.1.6 APC Initialization and Operating Considerations

As long as the APC timer (CPM timer 4) is not active, the APC parameters may be initialized in any order. The APC timer must be initialized last, following the initialization of the APC priority levels of all the ATM ports in the system. Failure to initialize the APC timer last will allow the scheduling algorithm to start prematurely, resulting in unpredictable behavior.

Furthermore, the APC scheduling table parameters must be initialized before any TRANSMIT CHANNEL ACTIVATE commands are issued. However, these commands may be issued at any time, whether the APC timer is active or inactive. For more information, see the description of the TRANSMIT CHANNEL ACTIVATE command.

Also, note that the physical interface (serial or UTOPIA) must be enabled and its associated clocks and synchronization signals must be active before the APC timer is activated. Otherwise, the transmit queue will simply overflow (causing an APCO interrupt).

39.1.7 Modifying Channel Transmit Pace

The APC pace parameter in the TCT of any channel can be manipulated at any time. Software can use this capability to support various dynamic traffic types, such as ABR.

39.1.8 Minimizing Cell Delay Variation

The number of cells sent in each time slot (determined by NCITS) has an effect on the maximum cell delay variation (CDV) since the order of the cells in a given time slot is not controlled. Therefore, the CDV increases as NCITS increases.

Also, as described before, the entries in the APC scheduling table are actually the heads of linked lists, which allows the APC to schedule multiple channels for the same time slot. These linked lists can be of unlimited depth, and although only NCITS cells are actually written to the transmit queue each time the APCT_PTRx advances, no cells are lost because the lagging APCT_SPTRx service pointer keeps track of the pending channels for next time. However, deeper linked lists result in more cell delay variation. Therefore, if cell delay variation is a concern, schedule channels such that they are distributed as uniformly as possible throughout the APC scheduling table. This can effectively be implemented by activating channels (using the TRANSMIT CHANNEL ACTIVATE command) at random intervals, such that they are not all written to the same APC scheduling table entry.

39.2 Direct Scheduling of Cells

The ATM controller implements an APC BYPASS command to allow the user to insert a channel number directly into the transmit queue on a cell-by-cell basis. This command can be used at any time in either serial mode or UTOPIA mode to insert a single cell for a

channel into the transmit queue, with no direct transmit queue pointer manipulation required.

If the APC is programmed to never schedule cells (either by disabling the APC timer or by removing all channels from the APC scheduling table with TRANSMIT CHANNEL DEACTIVATE commands), then groups of cells can be directly scheduled for back-to-back transmission. To send *n* cells back-to-back in serial mode, the user would write *n* channel numbers to the transmit queue and then advance TQAPTR by *n* entries. To send *n* cells back-to-back in UTOPIA mode the user would write (*n* - 1) channels numbers to the transmit queue, advance TQAPTR by (*n* - 1) entries, and then issue an APC BYPASS command for the *n*th cell.

39.3 Using the APC with Multiple ATM Ports

The APC algorithm always begins in page 4 of the dual-port RAM. For applications running only a single serial ATM port in either UTOPIA or serial mode, the page-4 APCST parameter should define the APC of page 4 as ‘inactive’ but still point to the next active APC, which is SCC1. The following discussion assumes that the APC for page 4 (the UTOPIA port) is active; however, the same general concepts apply to a system in which the APC for page 4 is inactive.

All the APCs are strobed by the same the APC timer request, which provides the common basic pace from which all the APCs (one for each ATM port configured) are scheduled. The APC timer activates several independent sub-timers in each APC implemented in the APCNT field of the APC parameters. Scheduling of cells from a particular APC will occur only when APCNT exceeds one. The APC timer should be programmed to supply an optimized common pace for all the APCs. This pace can then be divided or multiplied by setting NCITS appropriately for each APC.

For example, for a 50-MHz system implementing two SARs one of which is a 25-Mbps SAR and the other is a T1 serial SAR, the APC timer is programmed to generate an APC request every 4 cell times of the 25-Mbps SAR (CPM timer 4 should be programmed to 50 MHz/25 Mbps*53*8*4 = 3392). An NCITS value of 4 is selected for the 25-Mbps SAR and an NCITS value of 0.24 for the 1.544-Mbps SAR so that the 25-Mbps APC will schedule 4 cells per APC timer timeout, and the T1 APC will schedule one cell per 4.166 APC timer timeout (an average of 0.24) for the T1 SAR.

In the example shown in Figure 39-2, the APCST[NSER] field in page 4 points to page 1, APCST[NSER] of page 1 points to page 4. When an APC timer timeout occurs, the NCITS timer of page 1 is updated. Once per 4.166 requests the APCNT timer exceeds one, and APC1 schedules a channel for transmission. After the APCNT timer of page 1 is updated, APC2 is selected and schedules up to four (NCITS=4) channels.

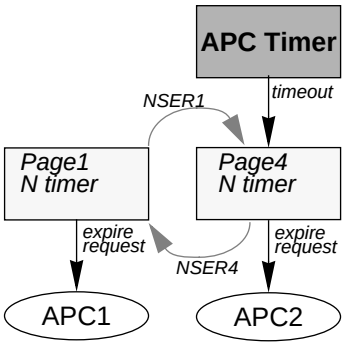


Figure 39-2. Example of Single PHY and Single Serial APC Configuration

Multi-PHY scheduling works in the same way as the scheduling process described above. Each PHY can support a different bit rate which is derived from the same APC timer basic rate. In this way, up to five different APCs (four multi-PHY and one serial mode SAR) can be supported. Upon an APC timer timeout, the CP begins servicing the serial APC and then services multi-PHY3 through multi-PHY0. The servicing order for the serial APC is defined by APCST[NSER]. Figure 39-3 shows the MPC855T configured with four multi-PHY SARs and one serial mode SAR

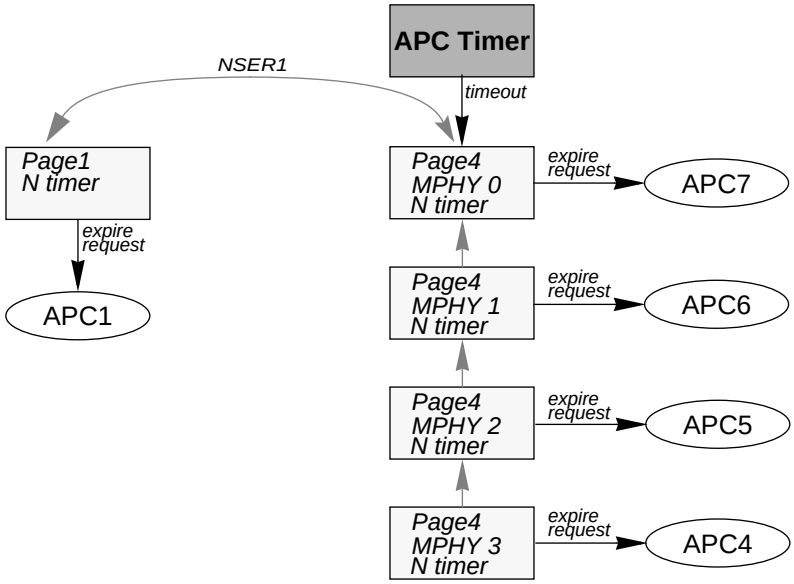


Figure 39-3. Example of Maximum Multi-PHY and Single-Serial APC Configuration

39.4 Using the APC Without Using UTOPIA

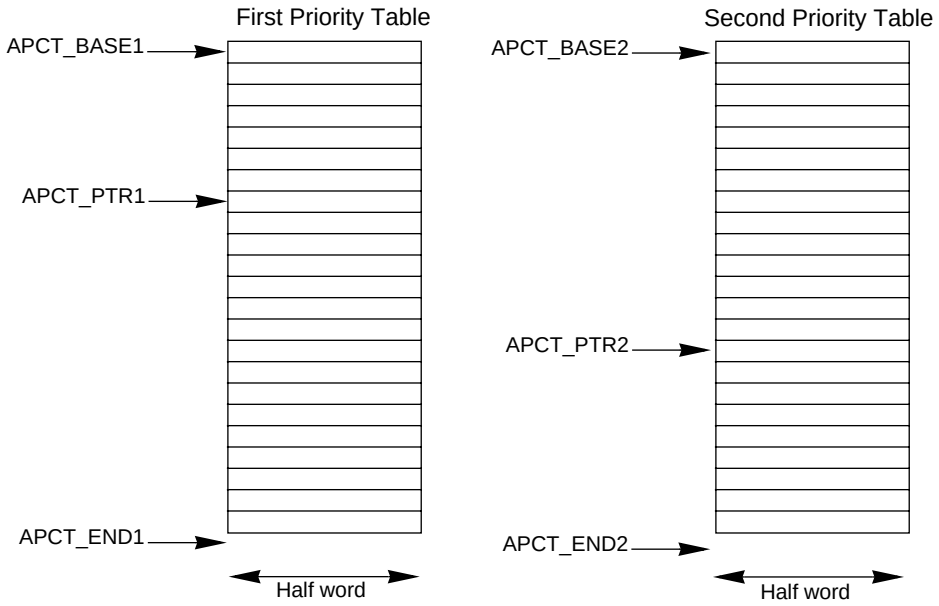
As described in Section 39.3, “Using the APC with Multiple ATM Ports,” the APC algorithm begins and terminates by referring to the APCST parameter on parameter page 4. Therefore, the APCST parameter on parameter page 4 must always be valid, even if the controller associated with parameter page 4 for the UTOPIA port, is not used in ATM mode.

If the UTOPIA port is not used, the APCST parameter on parameter page 4 must indicate that the APC on page 4 is disabled and must also point to the active APC page(s). This is accomplished using the APCST[NSER,CSER] mechanism described in Section 39.3, “Using the APC with Multiple ATM Ports.” The APC on parameter page 4 can be disabled by setting APCST[DIS].

39.5 APC Scheduling Tables

An APC scheduling table is a memory space located in the dual-port RAM. The user determines the number of entries in the APC scheduling table based on the required traffic parameters. The APC can be configured to handle two levels of priority through the configuration of APCST[PL2]. The APC first schedules channels from the first priority table, scheduling up to NCITS channels from the APCT_PTR1 slot. If there are fewer than NCITS channels in this slot and the PL2 bit is set, the APC tries to select the rest of the channels from the second priority table. The total number of channels selected from the tables is always NCITS or less. On each APC timeout, the APC advances the real-time pointers of both tables by one slot.

Note that during initialization all entries in the APC scheduling tables (from APCT_BASEx to APCT_ENDx) must be loaded with 0xFFFF (invalid channel number) to indicate an empty slot. The first- and second-priority APC scheduling tables are shown in Figure 39-4.



Note: APCT_END points to one position after the last entry in the table

Figure 39-4. APC Scheduling Tables

Note that the scheduling tables need not be the same length; the length of the table affects only the minimum programmable transmit rate, which might be different for high-priority or low-priority channels.

39.6 PHY Transmit Queues

The APC schedules up the NCITS channels in a time slot. It writes each channel number into a dedicated PHY transmit queue in the position indicated by TQAPTR. The transmitter's pointer TQTPTR, lagging behind the APC pointer, is used to read channel numbers. For each channel number read, the transmitter sends a single cell. The channels waiting to be sent lie between TQTPTR and TQAPTR. Figure 39-5 shows the organization of a PHY transmit queue.

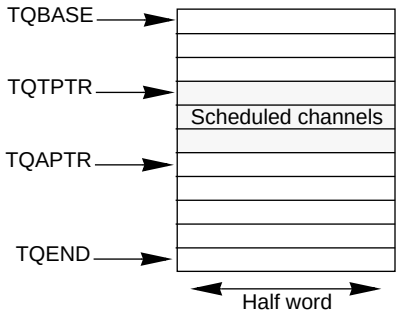


Figure 39-5. PHY Transmit Queue

A transmit queue never overflows because the TQAPTR pointer never wraps to point to the TQTPTR pointer. If the transmit queue is full, the APC does not insert more channels, and the APCT_SPTR stalls until space is available in the transmit queue. The depth of the transmit queue is equal to the number of entries minus 1.

39.7 APC Priority Levels

Table 39-1 describes the memory location and size of the user configurable parameters of the ATM pace controller.

Table 39-1. APC Priority Levels

Offset	Name	Width	Description	User Writes
0x00	**APCT_BASE1**	Half Word	APC scheduling table—First priority base pointer. See Table 39-2..	User defined
0x02	**APCT_END1**	Half Word	First APC scheduling table—Length. See Table 39-2..	User defined
0x04	**APCT_PTR1**	Half Word	First APC scheduling table pointer. See Table 39-2..	APCT_BASE1 value
0x06	**APCT_SPTR1**	Half Word	APC scheduling table first priority service pointer. See Table 39-2..	APCT_BASE1 value

Table 39-1. APC Priority Levels (continued)

Offset	Name	Width	Description	User Writes
0x08–0x0F	—		Reserved	-
0x10	**APC_MI**	Half Word	APC—Max iteration	User defined
0x12	**NCITS**	Half Word	Number of cells in time slot. See Table 39-2.	User defined
0x14	**APCNT**	Half Word	APC—N timer	0000
0x16-ox1F	—		Reserved	-
(n * 0x20) + 0x0	APCT_BASEn	Half Word	APC scheduling table base pointer for the N'th priority APC level	User defined
(n * 0x20) + 0x2	APCT_ENDn	Half Word	N'th table —Length	User defined
(n * 0x20) + 0x4	APCT_PTRn	Half Word	N'th APC scheduling table pointer	APCT_BASEn value
(n * 0x20) + 0x6	APCT_SPTRn	Half Word	N'th table APC service pointer	APCT_BASEn value‘
(n * 0x20) + 0x8 - (n * 0x20) + 0x3F	Reserved	—	—	—

Notes:

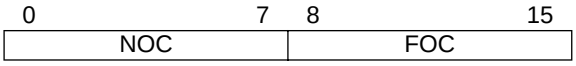
1. The offset is from APCPTR.
2. For single-PHY operation, the base address of the APC priority levels must be divisible by 32 (APCPTR must end with 0b00000).
3. Shaded area - Optional: Only used if APCST[PL2] bit is set.

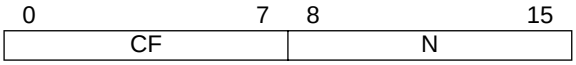
Table 39-2 describes the operation of the user-configurable APC priority level parameters.

Table 39-2. APC Priority Level Parameter Descriptions

Name	Description
APCT_BASEn	APC scheduling table base for the N'th priority APC level. Holds the pointer to the first entry in the APC scheduling table. The pointers should be half-word aligned (even address). APCPTR + (n * 0x20)
APCT_ENDn	APC scheduling table end for the N'th priority APC level. Holds the end pointer of the APC scheduling table, and is set to the address of the last entry in the APC scheduling table + 2 (APCT_END*n* = Last_Entry*n*+2).
APCT_PTRn	APC scheduling table pointer for the N'th priority APC level. Holds the location of the current APC time slot in the APC scheduling table. The APC advances the pointer on every APC N timer timeout. The APC scheduling table pointer should be initialized by the user to the APCT_BASEn value.
APCT_SPTRn	APC scheduling table service pointer for the N'th priority APC level. Used internally by the APC. The APCT_SPTRn parameter should be initialized by the user to the APCT_BASEn value.

Table 39-2. APC Priority Level Parameter Descriptions (continued)

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>NCITS</td><td>Number of cells in time slot. Parameter set by the user. It holds the number of cells which are transmitted in a time slot. This number can include fractions of a cell. The NCITS field is defined as follows:


The NCITS bit fields are described below.
<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>0–7</td><td>NOC</td><td>Number of cells. This field contains the integer value for the number of cells.</td></tr>
<tr><td>8–15</td><td>FOC</td><td>Fraction of cell. This field holds the fraction of cell where:
$NCITS = NOC + \frac{FOC}{256}$</td></tr>
</table></td></tr>
<tr><td>APCNT</td><td>APC N timer. Used internally by the APC. It should be initialized by the user to zero. The APC adds NCITS to this timer on every APC Timer request. This timer holds the APC N parameter and additional cell fraction remainder which is used by the APC. The APCNT field is defined as follows:


The APCNT bit fields are described below.
<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>0–7</td><td>CF</td><td>Cell fraction used by the APC.</td></tr>
<tr><td>8–15</td><td>N</td><td>APC N parameter.</td></tr>
</table></td></tr>
<tr><td>APC_MI</td><td>Max iteration. Number of times/steps that the APC advances the service pointer. This parameter limits the time spent in a single APC routine, thereby avoiding excessive APC latency. The recommended value for APC_MI is equal to the minimum value of TCT[APCP] (APC pace) of all channels, and should not exceed 32.</td></tr>
</table>

Chapter 40
ATM Exceptions

Interrupt handling for ATM channels involves two principle data structures: an event register (SCCE or IDSR1) and a circular ATM interrupt queue. The interrupt queue (one per controller) is shown in Figure 40-1.

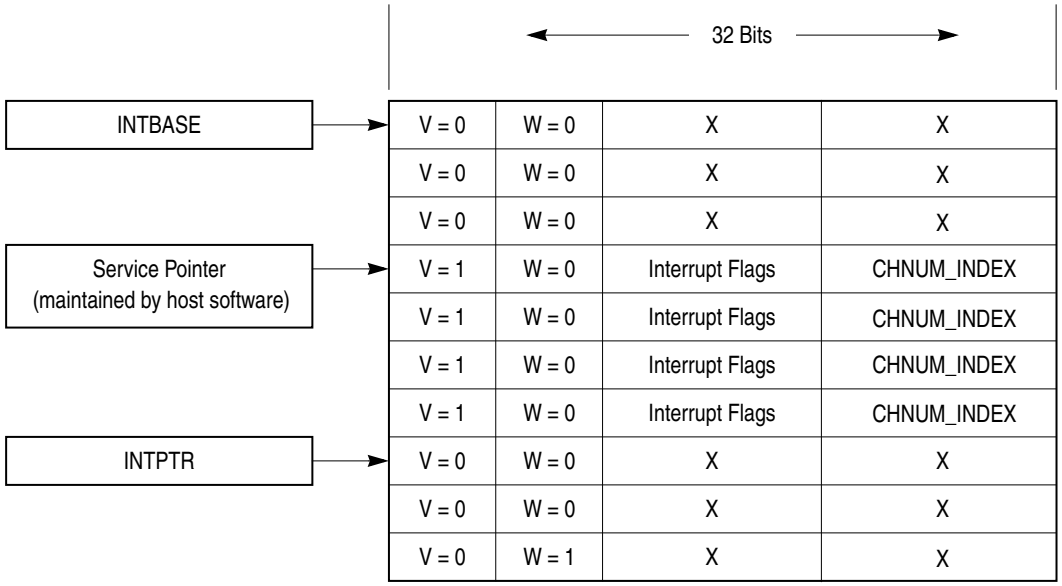


Figure 40-1. ATM Interrupt Queue

The INTBASE and INTPTR pointers are host-initialized global ATM parameters that respectively point to the starting location of the queue structure in external memory and to the current empty position available to the CP. The end of the queue is defined by the entry containing the wrap (W) bit set.

When an ATM channel generates an interrupt request, the CP writes a new entry to the queue consisting of the channel's code number and a description of the exception. The valid (V) bit is then set and INTPTR is advanced. After the CP writes the last entry in the queue (W bit set), it re-initializes INTPTR to point to the base of the queue (INTBASE).

For each event sent to an interrupt queue, the CP decrements a down counter which has been initialized to a threshold number of interrupts. When the counter reaches zero, the controller's global interrupt (SCCE[GINT] or IDSR1[GINT]) is set. The user controls how

often the host application is interrupted to service new entries in the queue by programming the interrupt threshold (INT_ICNT) in the parameter RAM; see Table 37-1.

After an interrupt request, the host's interrupt service routine polls the controllers' event registers (SCCE[GINT] and/or IDSR1[GINT]) to determine which controller is requesting service. After clearing GINT, the host processes each valid queue entry in turn, clearing each V bit and all flagged event bits so that the entry can be reused by the CP. The host continues servicing entries until it reaches an invalid entry (whose V bit is already cleared).

40.1 ATM Event Registers

The ATM event registers generate interrupts to the host and report on events that are common to all channels of a controller. The global interrupt (GINT) bit indicates that at least one channel-specific interrupt has been added to the interrupt queue. In UTOPIA mode, the IDSR1 register is used as an exception event register; in serial ATM mode the SCC event register (SCCE) is used for events.

40.1.1 UTOPIA Event Register (IDSR1)

The IDSR1 register is the ATM event register when operating in UTOPIA mode. IDSR1 is used to report events and generate interrupt requests for the UTOPIA interface. Note that in UTOPIA mode, interrupts from the ATM port are reported with an IDMA1 vector in the CIVR, and the IDMA1 bit is set in the CIPR. Setting the corresponding bit in the mask register IDMR1 enables the actual generation of the interrupt request. Event bits are cleared by writing ones; writing zeros has no effect. The UTOPIA event and mask registers are shown in Figure 40-2.

Bits	0	1	2	3	4	5	6	7
FIELD	—			SYNC	IQOV	GINT	—	—

Figure 40-2. UTOPIA Event Register (IDSR1) and Mask Register (IDMR1)

Table 40-1 describes the UTOPIA event register fields.

Table 40-1. UTOPIA Event Register (IDSR1) Field Descriptions

Bits	Name	Description
0–2	—	Reserved
3	SYNC	When this occurs the receiver stops receiving cells until it regains SOC synchronization. SRSTATE[SNC] indicates that the receiver is waiting for resynchronization. Note that the SYNC interrupt can be issued multiple times during the synchronization process until full synchronization is achieved.
4	IQOV	Interrupt queue overflow. Set by the CP whenever an overflow condition in the interrupt queue occurs. This condition occurs if the CP attempts to write a new interrupt entry into a valid entry (V = 1) not yet handled by the host.

Table 40-1. UTOPIA Event Register (IDSR1) Field Descriptions (continued)

Bits	Name	Description
5	GINT	Global interrupt. Indicates that at least one new entry has been added to the interrupt queue. After clearing the GINT event flag, the host begins processing the entries using the service pointer. The host returns from the interrupt handler when it reaches an invalid queue entry (V = 0).
6–7	—	Reserved

40.1.2 Serial ATM Event Register (SCCE)

The SCCE acts as the ATM event register for serial mode and is used to report events and generate interrupt requests. Note that in serial ATM mode, interrupts from the ATM port are reported with the SCC vector in the CIVR, and the SCC bit is set in the CIPR. Setting the corresponding bit in the mask register SCCM enables the actual generation of the interrupt request. Event bits are cleared by writing ones; writing zeros has no effect. Figure 40-3 shows the serial ATM event and mask registers.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	—			GLR	GLT	DCC	—					SYNC	IQOV	GINT	GUN	GOV

Figure 40-3. Serial ATM Event Register (SCCE) and Mask Register (SCCM)

Table 40-2 describes the serial ATM event register fields.

Table 40-2. Serial ATM Event Register (SCCE) Field Descriptions

Bits	Name	Description
0–2	—	Reserved
3	GLR	Glitch on receive. A clock glitch has been detected by the SCC on the receive clock.
4	GLT	Glitch on transmit. A clock glitch has been detected by the SCC on the transmit clock.
5	DCC	DPLL carrier sense status change. Indicates carrier sense status generated by the DPLL has changed state. The value of the DCC bit is valid only when the DPLL is enabled.
6–10	—	Reserved
11	SYNC	Cell synchronization changed status. Indicates that the receiver has lost or gained cell delineation. The SYNC interrupt is signaled whenever the receiver changes lock status (refer to the ASTATUS lock bit in Section 37.7, “Serial Cell Synchronization Status Register (ASTATUS).” If synchronization is lost (lock bit is cleared), the SYNC interrupt indicates a fatal ATM reassembly error because the affected channels are unknown. When this happens, the receiver stops receiving data from all channels and all data transfers to memory halt. After re-initializing the channels, the host may resume receiving cells by executing the RESTART RECEIVE command (see Section 38.3, “ATM Commands”) for each channel.
12	IQOV	Interrupt queue overflow. Set by the CP whenever an overflow condition in the interrupt queue occurs. This condition occurs if the CP attempts to write a new interrupt entry into a valid entry (V = 1) not yet handled by the host.

Table 40-2. Serial ATM Event Register (SCCE) Field Descriptions (continued)

Bits	Name	Description
13	GINT	Global interrupt. Indicates that at least one new entry has been added to the interrupt queue. After clearing the GINT event flag, the host begins processing the entries using the service pointer. The host returns from the interrupt handler when it reaches an invalid queue entry (V = 0).
14	GUN	Global transmitter underrun. Indicates that an underrun occurred in the SCC's transmitter FIFO. A GUN error is fatal because the affected channels are unknown. After GUN is set, the transmitter stops data transmission from all channels and sets the APC disabled status flag APCST[DIS]. The transmit line enters an idle state (logic high). After re-initializing the channels, the host may resume transmission by issuing a RESTART TRANSMIT command (see Section 38.3, "ATM Commands") for each channel. For a faster recovery from a GUN error, re-initialize TSTATE by writing STFCR to the first byte, clearing the second byte, and leaving the third and fourth bytes as is. Then the APC can be restarted by clearing APCST[DIS]. This procedure results in corrupted transmit frames initially. (TSTATE should normally be modified only during system initialization.) Note also that clearing APCST[DIS] may be overwritten by the APC scheduling process; therefore, the user should verify that APCST[DIS] has indeed been cleared after a minimum of 50 system clocks.
15	GOV	Global receiver overrun. Indicates that an overrun occurred in the SCC's receiver FIFO. A GOV error is fatal because the affected channels are unknown. After GOV is set, the receiver stops receiving data from all channels and halts all data transfers to memory. After re-initializing the channels the host may resume receiving by issuing a RESTART RECEIVE command (see Section 38.3, "ATM Commands") for each channel. For a faster recovery from a GOV error, re-initialize RSTATE by writing SRFCR to the first byte, clearing the second byte, and leaving the third and fourth bytes as is. This procedure initially results in corrupted receive frames which should be disposed of by software. (RSTATE should normally be modified only during system initialization.)

40.2 Interrupt Queue Entry

Each entry in the ATM interrupt queue contains event information for a specific ATM channel. During initialization, the host software should clear all queue entries and set the wrap bit (W) only for the last entry. The format of an interrupt queue entry is shown in Figure 40-4.

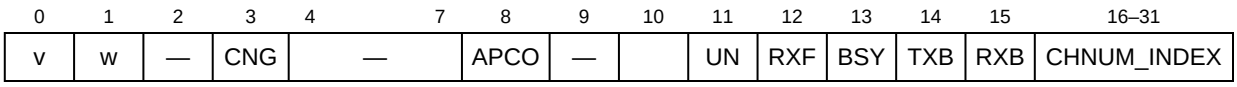


Figure 40-4. Interrupt Queue Entry

Table 40-3 describes the fields of an interrupt queue entry.

Table 40-3. Interrupt Queue Entry Field Descriptions

Bit	Name	Description
0	V	Valid bit. Indicates that this entry contains valid interrupt information. The CP sets this bit when generating a new entry. The V bit and all event bits should be cleared by the host service routine immediately after reading the entry.
1	W	Wrap bit. Indicates the last entry in the interrupt queue. After the CP writes to this entry, it moves to the beginning of the queue for the next event; that is, INTPTR is re-initialized to INTBASE. After the host services this entry, it should move to the beginning of the queue for the next entry to be processed; that is, the service pointer should be re-initialized to INTBASE. During initialization, the host should set the W bit only for the last entry of the queue.
2	—	Reserved
3	CNG	Congestion. Set by the CP when a congestion indication on a received cell (the middle bit of the PTI field is set). This interrupt applies only to channels whose RCT[CNGI] is set.
4–7	—	Reserved
8	APCO	APC overrun. Set by the CP if an APC scheduling table overruns. The address of the affected APC scheduling table is placed in the CHNUM_INDEX field. Indicates that the total programmed cell rate for this APC priority level is greater than the maximum cell rate capability of the transmitter. That is, the real-time scheduling pointer APCT_PTR has wrapped around to the service pointer APCT_SPTR position, causing an entire APC scheduling pass to be lost because the scheduling pointer begins overwriting the time slots with new channel numbers. Note however that no cells are lost, only the cell rates of the channels belonging to this scheduling table have been diminished. A scheduling table overrun occurs when (1) the programmed pace is greater than NCITS (that is, if NCITS = 1, and 1/APCP1 + 1/APCP2 + 1/APCP*n* > 1), (2) the APCT_SPTR has stalled for some reason, such as a full transmit queue, or (3) this level's APC_MI is too low relative to higher APC priority levels.
9 –10	—	Reserved
11	UN	Transmit underrun. Occurs for both AAL0 and AAL5 channels when scheduled to transmit without sufficient data to form a complete cell. No other action is taken and the channel remains enabled. If more data is not supplied, another UN exception is generated the next time the channel is scheduled. When an underrun occurs, an idle cell is sent, either generated by the MPC855T(in serial mode) or by the UTOPIA PHY (in UTOPIA mode).
12	RXF	Receive frame. Indicates that a complete AAL5 frame has been received.
13	BSY	Busy. Indicates that a cell was received but discarded due to lack of empty buffers. For AAL0, the receiver attempts again to open the same BD when the next cell for this channel arrives. For AAL5, the remaining cells of the current frame are discarded. After the last cell of the frame (PTI[1]=1) is received (but not stored), the receiver attempts again to open the same BD when the first cell of the next frame arrives.
14	TXB	Transmit buffer. Indicates the transmitter has sent the last cell of a buffer to the UTOPIA interface (when operating in UTOPIA mode) or to the serial FIFO (when operating in serial mode). This exception is enabled through the I bit in the TxBD.

Table 40-3. Interrupt Queue Entry Field Descriptions (continued)

Bit	Name	Description
15	RXB	Receive buffer. Indicates a buffer has been received. For AAL5, the buffer is not the last buffer in the frame (indicated by RXF). The RXB interrupt is also generated by other errors that occur when receiving; in this case, the error condition is reported in the RxBD. This exception is enabled through the I bit in the RxBD.
16–31	CHNUM_INDEX	Channel number index. This field represents the CH_CODE of the channel associated with this interrupt entry or the APC scheduling table base address experiencing overrun. When not operating in extended channel mode, the CHNUM_INDEX field contains the channel's RCT or TCT address in dual-port RAM. In extended channel mode, CHNUM_INDEX is the channel number. If the interrupt is an APC overrun (APCO is set), the CHNUM_INDEX field contains the dual-port RAM offset of the APC priority level experiencing the overrun.

40.3 Interrupt Queue Mask (IMASK)

IMASK is the interrupt mask for both the receive and transmit sides of a channel and is located in the RCT; see Section 36.2.1, "Receive Connection Table (RCT). Shown in Figure 40-5, it allows the user to enable or disable interrupt generation. If a bit in IMASK is cleared, the corresponding event does not cause an entry to be added to the interrupt queue, and the GINT global interrupt counter is not incremented.

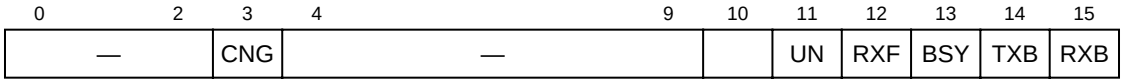


Figure 40-5. Interrupt Queue Mask (IMASK)

Note that because the masking is performed in microcode, approximately 40 system clocks must elapse for a change in IMASK to take effect.

Chapter 41
Interface Configuration

The following sections describe the programming of registers and parameters for ATM operations through both the UTOPIA and serial interfaces.

41.1 General ATM Registers

This section describes the general ATM registers.

41.1.1 Port D Pin Assignment Register (PDPAR)

The ATM and UT bits have been added to the PDPAR register, shown in Figure 41-1. The PDPAR register is cleared at system reset.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	ATM	UT	—	DD3	DD4	DD5	DD6	DD7	DD8	DD9	DD10	DD11	DD12	DD13	DD14	DD15
Reset	00			0_0000_0000_0000												
Oper	R/W															
ADDR	Offset to IMMR: 0x972 (PDPAR)															

Figure 41-1. Port D Pin Assignment Register (PDPAR)

The fields in the PDPAR register are described in Table 41-1.

Table 41-1. PDPAR Field Descriptions

Bits	Name	Description
0	ATM	ATM global enable. 0 =Disable ATM SAR functionality 1 =Enable ATM SAR functionality
1	UT	UTOPIA enable. Determines whether the parameter RAM's page 4 (SCC4) operates in serial or UTOPIA mode. 0 =Serial mode using page 4 1 = UTOPIA mode

Table 41-1. PDPAR Field Descriptions (continued)

Bits	Name	Description
2	—	Reserved
3–15	DDx	Signal assignment. Determines whether the signal is a general-purpose I/O signal or performs a dedicated function. 0 = General-purpose I/O. The peripheral functions of the signal are not used. 1 = Dedicated peripheral function. The signal performs the function assigned by the internal module.

41.1.2 APC Timer (CPM Timer 4)

The CPM general-purpose timer number 4 is used internally by the ATM pace controller as the APC timer for both serial ATM and UTOPIA modes. The APC timer should be programmed to run in active-low pulse and restart mode (see Section 17.2, “CPM General-Purpose Timers”). The APC timer period should be programmed according the required APC rate, which is discussed in Section 39.1, “APC Algorithm.”

41.1.3 RISC Timer

A dedicated RISC timer programmed by host software with the desired time-out interval can be used to implement a receiver time-out error check. The RISC timer is specified in TSTA (time-stamp timer address) in the ATM parameter RAM. See Section 18.7, “The RISC Timer Table,” for additional information.

41.2 UTOPIA Mode Registers

When operating in UTOPIA mode the PHY layer is connected to the MPC855T UTOPIA interface. The UTOPIA data signals and some of the control signals are connected to port D. The remaining UTOPIA control signals are connected to ports B and C. The UTOPIA mode requires several registers to be configured as described in the following sections.

41.2.1 System Clock Control Register (SCCR)

The system clock control register is described in Section 14.6.1, “System Clock and Reset Control Register (SCCR).” SCCR[27–31] control the UTOPIA clock (UTPCLK).

The frequency of the UTPCLK defaults to system frequency. The frequency ratio between the system clock and UTPCLK is an integer value ($freq_{sys}/freq_{utopia}$ = integer > 0). The UTOPIA clock has a 50% duty cycle and is derived from the system frequency divided by two dividers. Note that the UTOPIA clock must be programmed to operate at a frequency less than or equal to 25 MHz. The SCCR[DFUTP] and SCCR[DFAUTP] fields should be programmed such that the total UTOPIA clock division factor never exceeds 5 (that is, the

bounds of UTPCLK are 25 MHz > UTPCLK > SYSCLK/5). The UTOPIA clock frequency can be determined using the following formula:

$$\text{FREQutopia} = \frac{\text{FREQsys}}{(2^{\text{DFUTP}}) \times (2 \times \text{DFAUTP} + 1)}$$

For example, to achieve a 25-MHz UTOPIA clock with a 50-MHz system clock, DFUTP should be programmed to 0b001, and DFAUTP should be cleared. The SCCR is shown in Figure 41-2.

BIT	0	1–26	27	28	29	30	31
FIELD	as described in Section 14.6.1, “System Clock and Reset Control Register (SCCR).”		DFUTP			DFAUTP	
RESET			0_0000				
OPER			R/w				
ADDR	Offset to IMMR: 0x280 (SCCR)						

Figure 41-2. System Clock Control Register (SCCR)

The fields related to the UTOPIA clock in the SCCR register are described in Table 41-2.

Table 41-2. SCCR Field Descriptions for the UTOPIA Clock

Bits	Name	Description
27–29	DFUTP	Division factor for UTPCLK. Divide the system clock by 2^{DFUTP}. The system clock division factor is limited to 4.
30–31	DFAUTP	Additional division factor for UTPCLK. Divide the system clock by (2 x DFAUTP + 1) 00 =Divide by 1 01 =Divide by 3 10 =Divide by 5 11=Reserved

41.2.2 Port B—TxClav

Port B includes the TxClav input signal. PB15 is configured to support the TxClav signal when PBPAR[15] is set and PBDIR[15] is cleared.

UTOPIA MPHY operations use the port B PHREQ[0–1] and PHSEL[0–1] signals. These signals are configured by clearing PBPAR[16–21] and PBDIR[16–17], and setting PBDIR[20–21].

41.2.3 Port C—RxClav Signal

In UTOPIA mode (PDPAR[UT] = 1), the port C PC15 pin provides the UTOPIA RxClav input signal. The PCPAR and PCDIR fields must be cleared and the PCSO field must be set to enable the RxClav signal input.

41.2.4 Port D—UTOPIA Data and Control Signals

Port D includes the UTOPIA data and control signals. When PDPAR[UT] is set, most of the port D signals are configured to support UTOPIA signals as shown in Table 41-3. The UTOPIA interface is described in Chapter 42, “UTOPIA Interface.”

NOTE

Port D must be initialized before Port C to prevent the CPM from trying to use IDMA functionality.

Signal	PDPAR = 0	PDPAR=1			Input to On-Chip Peripherals
		UT=0		UT=1	
		PDDIR=0	PDDIR=1		
PD15	Port D15	L1TSYNCA	MII-RXD3 (I)	UTPB[0]	L1TSYNCA=GND
PD14	Port D14	L1RSYNCA	MII-RXD2 (I)	UTPB[1]	L1RSYNCA=GND
PD13	Port D13	—	MII-RXD1 (I)	UTPB[2]	—
PD12	Port D12	—	MII-MDC (O)	UTPB[3]	—
PD11	Port D11	—	MII-TX-ERR (O)	RxEnb	—
PD10	Port D10	—	MII-RXD0 (I)	TxEnb	—
PD9	Port D9	—	MII-TXD0 (O)	UtpClk	—
PD8	Port D8	—	MII-RX_CLK (I)	—	—
PD7	Port D7	—	MII-RX-ERR(I)	UTPB[4]	—
PD6	Port D6	—	MII-RXDV (I)	UTPB[5]	—
PD5	Port D5	—	MII-TXD3 (O)	UTPB[6]	—
PD4	Port D4	—	MII-TXD2 (O)	UTPB[7]	—
PD3	Port D3	—	MII-TXD1 (O)	SOC	—

Table 41-3. Port D Pin Assignment

41.2.5 RISC Controller Configuration Register (RCCR)

The RCCR[DR1M,DR0M] bits must be set (level-sensitive IDMA request signals) to enable UTOPIA operation. Also, program RCCR[DRQP] to 0b01 to give SCC transfers higher priority.

41.2.6 UTOPIA Mode Initialization

The following procedure is required for proper initialization of the UTOPIA interface:

1. Because the UTOPIA port activates immediately upon initialization, configure the ATM parameters and data structures first.

1. Set SRFCR[DIS] to 1 to mask the RxClav signal.
2. Program PBPAR and PBDIR to enable TxClav.
3. Program PCPAR, PCDIR, and PCSO to enable RxClav.
4. Program PBPAR and PBDIR to enable TxClav.
5. Clear SRFCR[DIS] to unmask the RxCav signal, thereby enabling the UTOPIA interface. At least 20 system clocks must elapse between the configuration of the PDPAR and clearing SRFCR[DIS].

The ATM controller starts searching for SOC and sets SRSTATE[SNC] as soon as the first SOC is found.

41.3 Serial ATM Configuration

This section describes the configuration of registers for serial ATM operation.

41.3.1 RISC Controller Configuration Register (RCCR)

The RCCR[DR1M] bit must be set, and the RCCR[DRQP] field must be programmed to 0b01 to allow a higher priority for SCC transfers.

41.3.2 SCC Configuration for Serial ATM

To enable the SCC to operate in serial ATM mode, configure the SCC for transparent operation (because GSMR_L[MODE] has no explicit ATM option) and clear MRBLR in the SCC's parameter RAM. If MRBLR is programmed with a non-zero value, the SCC operates in transparent mode.

Be sure to initialize the ATM parameters and data structures before enabling serial ATM operation because transfers begin as soon as the SCC is enabled in the GSMR.

The following sections describe the programming of the SCC registers for serial ATM operation.

41.3.2.1 General SCC Mode Register (GSMR)

To configure the SCC as an ATM controller, program GSMR_H[TRX, TTX, CDP, CTSP, CDS, CTSS] (see Section 21.2.1, "General SCC Mode Register (GSMR)"). When the initialization sequence has been completed, GMSR_L[ENR, ENT] must be set to enable receive and transmit functions (see Section 21.4.3, "SCC Initialization").

The SCC running serial ATM does not support mixed mode operation (in which the SCC transmitter is configured for ATM transmissions and the receiver is in transparent mode, or vice versa).

41.3.2.2 Serial ATM Mode Register (PSMR)

The protocol-specific mode register (PSMR), shown in Figure 41-3, functions as the serial ATM mode register and controls both the scrambling and the HEC coset functions for the transmitter and receiver.

BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIELD	—	—	SCRAM	—	—	—	—	COSET	—	—	—	—	—	—	—	—
RESET	0000_0000_0000_0000															
OPER	R/W															
ADDR																

Figure 41-3. Serial ATM Mode Register (PSMR)

Table 41-4 describes the PMSR serial ATM fields.

Table 41-4. PSMR Serial ATM Field Descriptions

Bits	Name	Description
0–1	—	Reserved
2	SCRAM	Scrambling function during sending and receiving 0 = Disable cell payload scrambling. 1 = Enable cell payload scrambling.
3–6	—	Reserved
7	COSET	HEC coset function 0 = Do not apply the HEC coset rules. 1 = Apply the HEC coset to all cells sent and received.
8–15	—	Reserved

41.3.3 SI Configuration for Serial ATM

The serial interface (SI) can be configured to support the SCC running serial ATM using either the dedicated SCC pins (non-multiplexed serial interface—NMSI) or time-division multiplexing (TDM) through the time-slot assigner (TSA). (See Section 20.2, “The Time-Slot Assigner (TSA),” or Section 20.3, “NMSI Configuration.”)

Note that a serial ATM port using the TSA can be connected to E1, T1, and xDSL line interface devices.

Chapter 42
UTOPIA Interface

The MPC855T supports single- and multi-PHY ATM operations through an industry-standard UTOPIA interface. The sections below describe the signals provided for UTOPIA support, and signal timing for single- and multi-PHY ATM operations.

42.1 UTOPIA Single-PHY

The MPC855T acts as an ATM layer UTOPIA master per the ATM Forum UTOPIA level 1 specification for an ATM single-PHY configuration. The MPC855T implements the UTOPIA interface as an 8-bit wide bidirectional data bus using a cell-level handshake, and operates at frequencies up to 25 MHz. The UTOPIA controller controls all interface signals.

Assertion of transmit cell available (TxCav) or receive cell available (RxCav) issues a request to the CP to handle a receive or transmit operation. During the cell transfer, the UTOPIA controller controls the enable signals ($\overline{\text{TxEnb}}$ or $\overline{\text{RxEnb}}$) and the transmit start of cell signal (TxSOC). It also samples the RxSOC signal during the cell transfer.

Most of the UTOPIA signals are multiplexed on MPC855T port D pins as shown in Figure 42-1. The IDMA request connected to the DREQ0 signal is replaced with RxClav. TxClav signal is connected to the port C[12] signal.

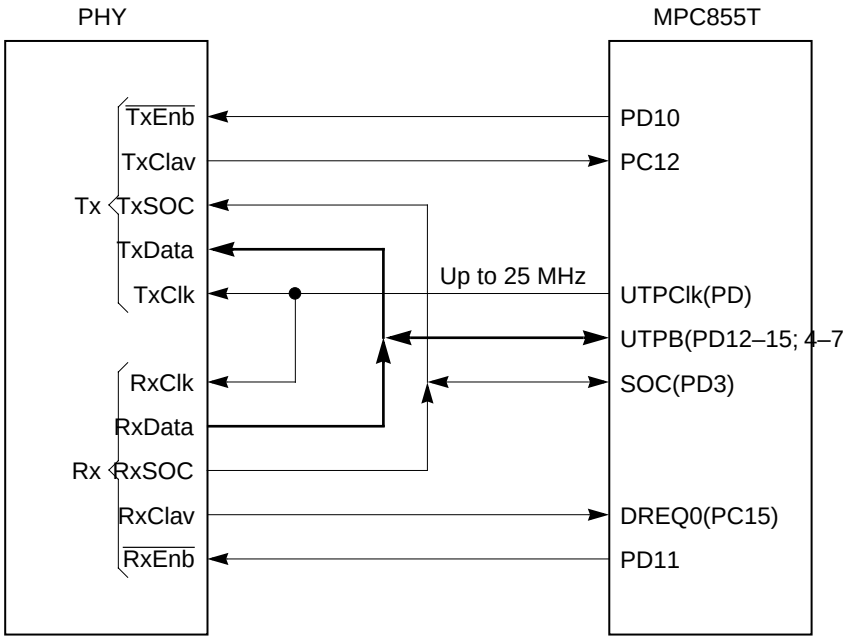


Figure 42-1. MPC855T UTOPIA Interface

The MPC855T implements a cell level interface. The cell level handshake is identical to the octet level handshake except that once the TxClav or RxClav signals are asserted the PHY must be capable of receiving or transmitting a whole cell. The MPC855T transmits or receives a whole cell directly to or from system memory during a receive or transmit operation.

42.1.1 Receive Cell Transfer Operation

Assertion of RxClav generates a request to receive a cell transfer. The MPC855T UTOPIA provides the cell level handshake, and as soon as RxClav is asserted, the PHY must be able to transfer a whole cell upon $\overline{\text{RxEnb}}$. The MPC855T's UTOPIA controller divides the cell transfers into 1 to 4 byte groups and uses $\overline{\text{RxEnb}}$ to control the transfer. For example, a 53-byte cell transfer sequence is divided into the following UTOPIA transfers:

- Header transfer (4 octets)
- UDF (HEC) transfer (1 octet)
- 12 cell body transfers (12 x 4 octets)

The MPC855T asserts $\overline{\text{RxEnb}}$ for each transfer. The following cycle the MPC855T starts sampling the UTOPIA bus (through the UTPB signals) and RxSOC. During the UDF tenure only one octet is transferred; all other sections are four octets long.

The receive start of cell timing sequence is shown in Figure 42-2. The circles shown during the data tenure represent the sampling points of the MPC855T. Note that RxClav is not sampled during the transfer.

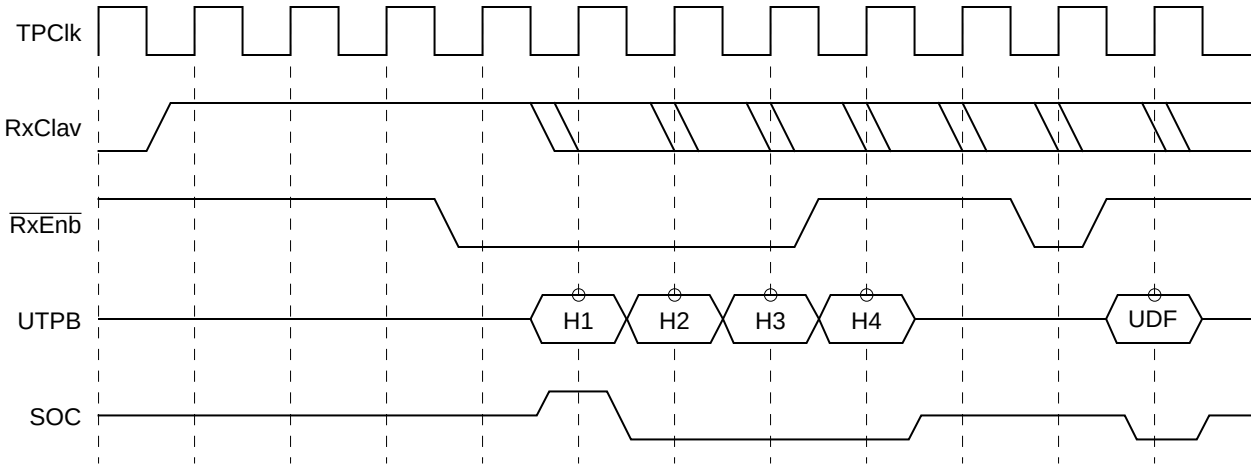


Figure 42-2. UTOPIA Receiver Start of Cell

The end-of-cell transfer timing sequence is shown in Figure 42-3. In this example the PHY was not ready with additional data and therefore deasserted RxClav. A few clocks later the PHY asserted RxClav again to indicate that data was available. If the PHY is ready to send additional data at the end of the current data tenure, the PHY can assert RxClav at any time during the data transfer and hold RxClav asserted until the first transfer the following data tenure.

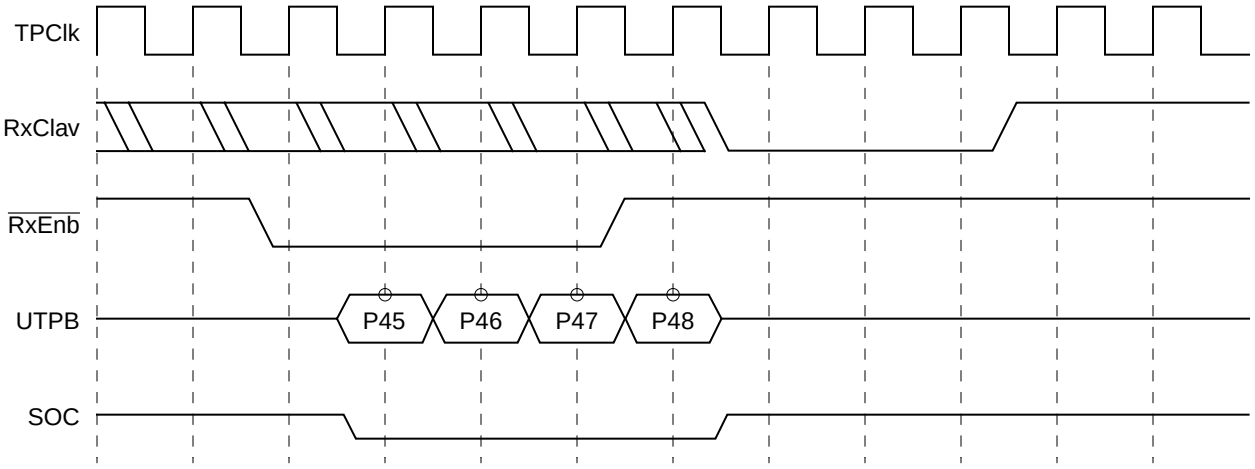


Figure 42-3. UTOPIA Receiver End of Cell

42.1.2 Transmit Cell Transfer Operation

Assertion of the TxClav signal generates a request for a cell to transmit. The MPC855T's UTOPIA interface implements the cell level handshake, and the PHY must be able to receive a whole cell upon assertion of the $\overline{\text{TxEnb}}$ signal. Note that a RxClav signal assertion generates a request that has higher priority than those caused by the assertion of the TxClav signal; transmit cell requests are granted by the CP when it has completed all the cell

receive requests. The MPC855T's UTOPIA controller divides the cell transfers into 1- to 4-byte groups and uses the $\overline{\text{TxEnb}}$ signal to control the transfer. For example, a 53-byte cell transfer sequence is divided into the following UTOPIA transfers:

- Header transfer (4 octets)
- UDF (HEC) transfer (1 octet)
- Cell payload transfers (total of 48 octets)

The MPC855T asserts $\overline{\text{TxEnb}}$, TxSOC, and TxPrty signals, and drives the UTOPIA bus (through the UTPB signals) with data for each transfer. The cell header is transferred in a 4-octet group. The UDF is transferred as a single octet. The cell payload is transferred in groups of 1 to 4 octets. The UDF (HEC) is not generated by the MPC855T; during the UDF transfer, the UTPB signals will be driven with 0x00.

42.1.2.1 UTOPIA Bus and SOC Drive

The UTOPIA bus (through the UTPB signals) and SOC signal are driven by the MPC855T only when $\overline{\text{TxEnb}}$ is asserted during transmit transfers; UTPB and SOC are three-stated when $\overline{\text{TxEnb}}$ is not asserted.

The transmit start-of-cell sequence is shown in Figure 42-4. Note that the TxClav signal is not sampled during the cell transfer.

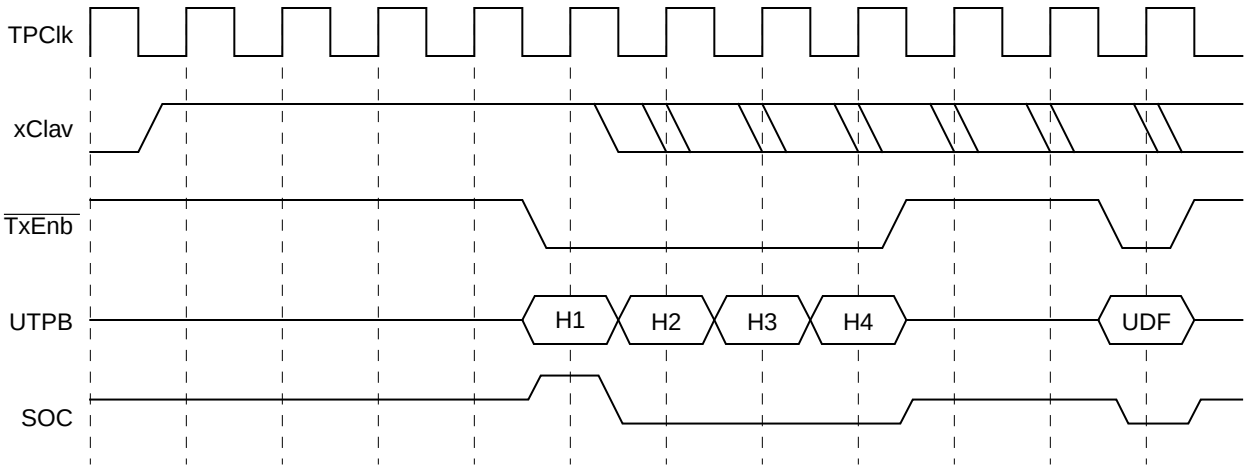


Figure 42-4. UTOPIA Transmitter Start of Cell

The transmit end-of-cell sequence is shown in Figure 42-5. In this example the PHY is ready for the next cell transfer and has asserted TxClav immediately at the end of the cell transfer. The TxClav signal should be kept asserted until the $\overline{\text{TxEnb}}$ signal is asserted by the MPC855T to indicate the transfer start of a new cell.

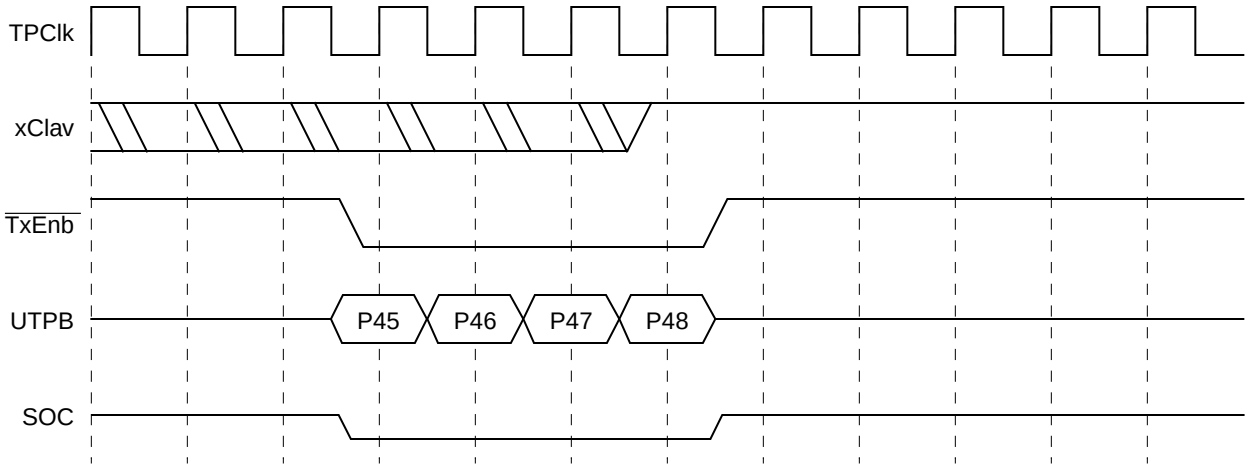


Figure 42-5. UTOPIA Transmitter End of Cell

42.2 UTOPIA Multi-PHY Operations

The MPC855T supports a multi-PHY interface through the use of PHY addressing signals. The following are guidelines for Multi-PHY operation:

- Up to 4 PHYS may be supported.
- Supports using additional PHY addressing signals - PHREQ and PHSEL.
- PHREQ indicates the receive requesting (available) PHY (i.e., PHREQ identifies the PHY requesting that the MPC855T receive its data). PHSEL is used to select the PHY that is to receive or transmit data.
- For the transmit side, Upon TxClav the MPC855T may chose to deliver the next cell to any of the available PHYs. Therefore, TxClav should be asserted only when **all** the PHYs are available.

¥ If additional port B pins are used as a general purpose output pins, a dedicated semaphore (PBF) should be used to prevent collision between Host and CP writes. (see PBF bit in STSTATE parameter ram field).

¥ PHSEL and PHREQ timing is with reference to the system clock.

42.2.1 Setting up PHSEL and PHREQ Pins

To drive a MPHY address, PHREQ (driven through the PB16 and PB17 signals) and PHSEL (driven through the PB20 and PB21 signals) must be set by the user to be general purpose signals. PHREQ should be programmed as input signals and PHSEL programmed as output signals through the associated bits in the PBPAR and PBDIR registers. Both PHSEL and PHREQ are read or written directly through port B synchronously with the

system clock; refer to Section 33.3, “Port B” for additional information about system clocking and port read and write operations.

Program PBPAR pins PB16, PB17, PB20 and PB21 to zero, that is, as general purpose I/O pins. Then program PBDIR pins PB16 and PB17 to zero (as inputs) and PBDIR pins PB20 and PB21 to one (as outputs). PB16 and PB17 will be used for PHREQ where PB16 is the most significant bit, PB20 and PB21 will be used for PHSEL where PB19 is the most significant bit.

42.2.2 Receive Cell Transfer Operation

The multi-PHY UTOPIA cell transfer protocol is basically the same as that used for single-PHY transfers, except that prior to the assertion of $\overline{\text{RxEnb}}$ the MPC855T reads the PHY address through PHREQ. The PHY address is then written to PHSEL for use during the address match process.

The MPC855T RxClav input pin may be asserted together with a valid PHY number on the PHSEL pins. The assertion of RxClav will generate a request to the CP. As soon as the request is accepted by the CP (after at least several UTOPIA clocks) PHREQ will be read by the CP. If PBF bit is 0, the MPC855T will write the selected PHY# to the PHSEL pins and only after several UTOPIA clocks the RxEnb will be asserted and cell reception will start. If PBF is set, the CP will keep reading PHREQ, waiting for PBF to be cleared by the HOST (see PBF bit in STSTATE parameter ram field).

PHREQ must remain valid from the assertion of RxClav until $\overline{\text{RxEnb}}$ is asserted. PHSEL is driven by the MPC855T and remains valid from the cycle the MPC855T has selected a PHY until the end of the cell transfer through the UTOPIA interface, as shown in Figure 42-6.

NOTE

During RxCav assertion, PHSEL still may be changed dynamically if a higher priority PHY asserted its cell available pin. The MPC855T PHSEL will be sampled at the rising edge of the system clock.

42.2.3 Transmit Cell Transfer Operation

When transmitting, the MPC855T writes the PHY address to PHSEL to select the currently addressed PHY prior to the assertion of $\overline{\text{TxEnb}}$. The MPC855T TxCav input pin may be asserted only when all the PHYs are ready to accept a cell. The assertion of TxClav will generate a request to the CP. As soon as the request is accepted by the CP, PHSEL will be updated to select the current PHY and only after several UTOPIA clocks the $\overline{\text{TxEnb}}$ will be asserted and cell transfer will start.

Note that requiring TxCav to be asserted only when all PHYs are ready to receive a cell does not mean that the transmission rate is governed by the slowest PHY. The scheduler (APC using PHY programmed transmission rate) will place more cells in the transmit queue for faster PHYs. The transfers and subsequent assertions of TxClav by the PHYs are essentially instantaneous because the UTOPIA interface transfer is faster compared to the line rate and the use of cell FIFOs by the PHYs. If any of the PHYs malfunction (die) all transmission of cells is halted since TxClav will not be asserted. External logic is needed to detected and correct this condition and deactivation of the impacted channels should be carried out by the application software.

PHSEL is driven by the MPC855T and remains valid starting from the cycle the MPC855T has selected a PHY until the end of the cell transfer through the UTOPIA interface, as shown in Figure 42-8.

42.2.4 Example MPHY Implementation

Figure 42-6 is an overview example for implementing a Multi PHY system using the MPC855T. The TxClav will be asserted when all PHYS are ready to accept a cell. PhSel decode is used to select the PHY several clocks before TxEnb or RxEnb are asserted. The priority mux (Priority encode) is a “box” which prioritizes the PHYs in the Rx direction. RxClav should be asserted when at least one of the PHYs is ready to deliver a cell.

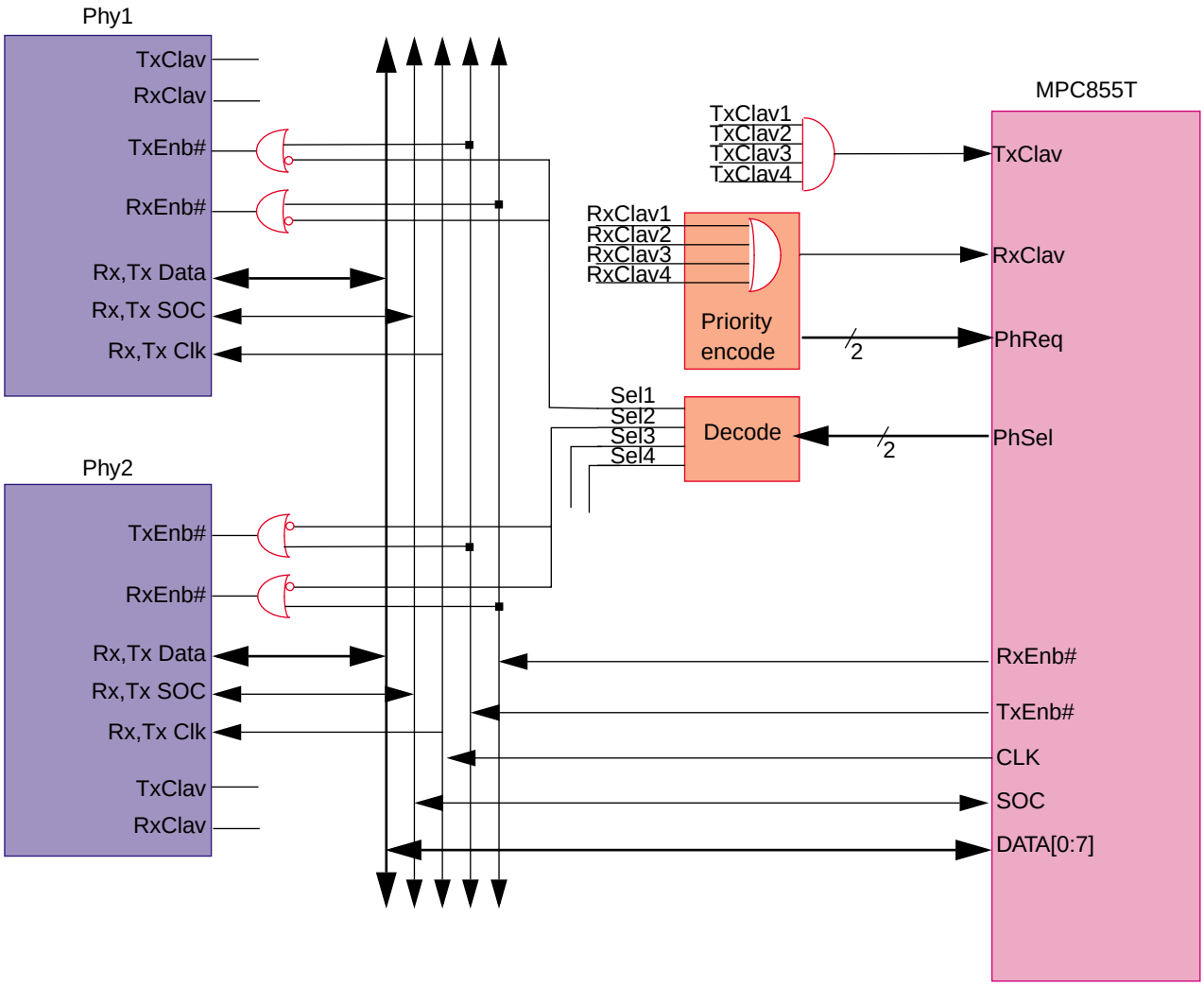


Figure 42-6. Multi-PHY Implementation Example

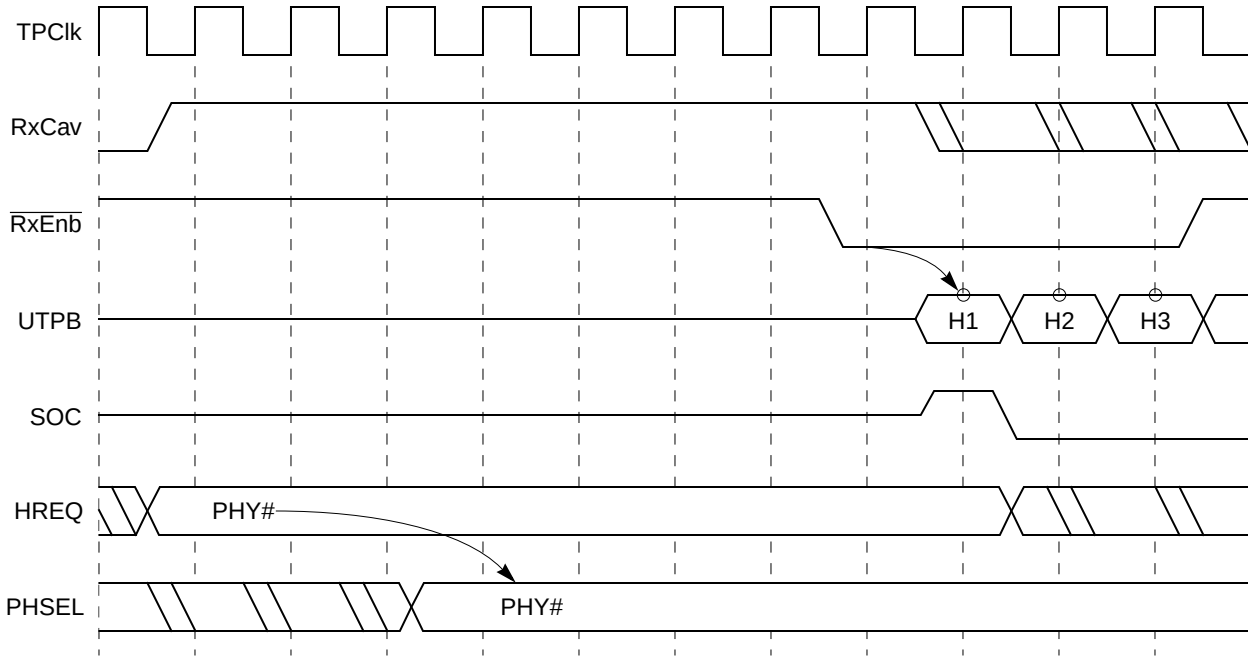


Figure 42-7. UTOPIA Receiver Multi-PHY Example

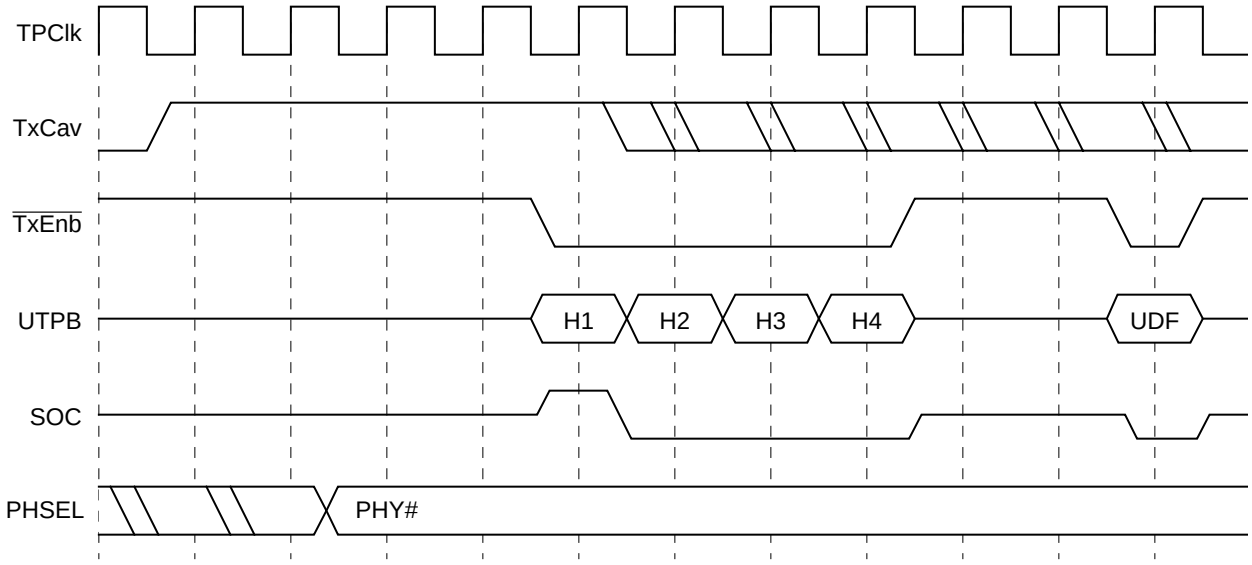


Figure 42-8. UTOPIA Transmitter Multi-PHY Example

42.3 UTOPIA Interface Transfer Timing

Table 42-1 describes the UTOPIA interface timing for data transfers and receive and transmit transitions.

Table 42-1. UTOPIA Interface Transfer Timing

UTOPIA Transfer Type	Clock Ratio	Transition Time
Interval between $\overline{\text{RxEnb}}$ negation and $\overline{\text{RxEnb}}$ assertion, or $\overline{\text{TxEnb}}$ negation and $\overline{\text{TxEnb}}$ assertion for successive data blocks (payload bytes 5-8 and 9-12).	UTPCLK/SYSCLK = 1	8 UTPCLKs minimum
	UTPCLK/SYSCLK = 1/2	4 UTPCLKs minimum
	1/3 < UTPCLK/SYSCLK < 1/5	0-3 UTPCLKs minimum
Interval between $\overline{\text{RxEnb}}$ negation and $\overline{\text{TxEnb}}$ assertion.	—	50 SYSCLKs minimum
Interval between $\overline{\text{TxEnb}}$ negation and $\overline{\text{RxEnb}}$ assertion.	—	20 SYSCLKs minimum
Interval between $\overline{\text{RxEnb}}$ negation and subsequent $\overline{\text{RxEnb}}$ assertion for header transfers.	—	6 SYSCLKs minimum
Interval between $\overline{\text{TxEnb}}$ negation and subsequent $\overline{\text{TxEnb}}$ assertion for header transfers.	—	4 SYSCLKs minimum

Part VII
Fast Ethernet Controller (FEC)

Intended Audience

Part VII is intended for system designers who need to use the MPC855T fast ethernet capabilities. It assumes a basic understanding of the PowerPC exception model, the MPC855T interrupt structure, as well a working knowledge of 10/100 base-T Ethernet. A complete discussion of fast ethernet is beyond the scope of this book.

Contents

Part VII contains the following chapter:

- Chapter 43, “Fast Ethernet Controller (FEC),” describes the Fast Ethernet controller. It provides general descriptions of supported operations, full descriptions of the supporting registers, and initialization information.

Conventions

This document uses the following notational conventions:

Bold	Bold entries in figures and tables showing registers and parameter RAM should be initialized by the user.
mnemonics	Instruction mnemonics are shown in lowercase bold.
italics	Italics indicate variable command parameters, for example, **bcctr***x*. Book titles in text are set in italics.
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote binary number
rA, **r**B	Instruction syntax used to identify a source GPR
rD	Instruction syntax used to identify a destination GPR
REG[FIELD]	Abbreviations or acronyms for registers or buffer descriptors are shown in uppercase text. Specific bits, fields, or numerical ranges

	appear in brackets. For example, MSR[LE] refers to the little-endian mode enable bit in the machine state register.
x	In certain contexts, such as in a signal encoding or a bit field, indicates a don’t care.
n	Indicates an undefined numerical value
¬	NOT logical operator
&	AND logical operator
|	OR logical operator

Acronyms and Abbreviations

Table i contains acronyms and abbreviations used in this document.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
BD	Buffer descriptor
CPM	Communication processor module
CRC	Cyclic redundancy checking
DMA	Direct memory access
FEC	Fast Ethernet Controller
FIFO	First-in/first-out (buffer)
MAC	Media access control
MII	Media-independent interface
Rx	Receive
SFD	Start frame delimiter
SIU	System interface unit
Tx	Transmit

Chapter 43
Fast Ethernet Controller (FEC)

This chapter describes the FEC as specifically implemented on the MPC855T. The 10/100 Fast Ethernet controller with integrated FIFOs and bursting DMA is implemented independently, so high-performance Fast Ethernet connectivity can be achieved without affecting the CPM performance.

43.1 Features

The following sections summarize key FEC features.

- 10/100 base-T support
  - Full compliance with the IEEE 802.3u standard for 10/100 base-T
  - Support for three different physical interfaces: 100-Mbps 802.3 media-independent interface (MII), 10-Mbps 802.3 MII, and 10-Mbps 7-wire interface
  - Large on-chip transmit and receive FIFOs to support a variety of bus latencies
  - Retransmission from the transmit FIFO after a collision
  - Automatic internal flushing of the receive FIFO for runts and collisions
  - External BD tables of user-definable size allow nearly unlimited flexibility in management of transmit and receive buffer memory
- 10/100 base-T media access control (MAC) features
  - Address recognition for broadcast, single station address, promiscuous mode, and multicast hashing
  - Full MII support
  - Interrupts supported per frame or per buffer (selectable buffer interrupt functionality using the I bit is not supported however.)
  - Automatic interrupt vector generation for receive and transmit events (Tx interrupts, Rx interrupts, and non-time critical interrupts)
  - Ethernet channel uses DMA burst transactions to transfer data to and from external memory

43.1.1 FEC Block Diagram

The FEC, the embedded MPC8xx core, the system interface unit (SIU), and the CPM all use the 32-bit internal bus. Figure 43-1 is a block diagram of the FEC.

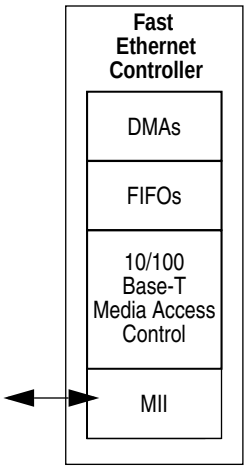


Figure 43-1. FEC Block Diagram

The FEC complies with the IEEE 802.3 specification for 10- and 100-Mbps connectivity. Full-duplex 100-Mbps operation is supported at system clock rates of 40 MHz and higher. A 25-MHz system clock supports 10-Mbps operation or half-duplex 100-Mbps operation.

The implementation of bursting DMA reduces bus usage. Independent DMA channels for accessing BDs and transmit and receive data minimize latency and FIFO depth requirements.

Transmit and receive FIFOs further reduce bus usage by localizing all collisions to the FEC. Transmit FIFOs maintain a full collision window of transmit frame data, eliminating the need for repeated DMA over the system bus when collisions occur. On the receive side, a full collision window of data is received before any receive data is transferred into system memory, allowing the FIFO to be flushed in the event of a runt or collided frame, with no DMA activity. However, external memory for buffers and BDs is required; on-chip FIFOs are designed only to compensate for collisions and for system bus latency.

Independent TxBD and RxBD rings in external memory allow nearly unlimited flexibility in memory management of transmit and receive data frames. External memory is inexpensive, and because BD rings in external memory have no inherent size limitations, memory management can be easily optimized to system needs.

43.2 Fast Ethernet Controller Operation

This section discusses the operation of the FEC.

43.2.1 Transceiver Connection

The FEC supports both an MII interface for 10/100 Mbps Ethernet and a seven-wire serial interface for 10-Mbps Ethernet. The interface mode is selected by R_CNTRL[MII_MODE], described in Section 43.4.1.20, “Receive Control Register (R_CNTRL).” Table 43-1 shows the 18 MII interface signals that are defined by the 802.3 standard.

Table 43-1. MII Signals

Signal Description	FEC Signal Name
Transmit clock	TX_CLK
Transmit enable	TX_EN
Transmit data	TXD[3:0]
Transmit error	TX_ER
Collision	COL
Carrier sense	CRS
Receive clock	RX_CLK
Receive enable	RX_DV
Receive data	RXD[3:0]
Receive error	RX_ER
Management channel clock	MDC
Management channel serial data	MDIO

Serial-mode connections to the external transceiver are shown in Table 43-2.

Table 43-2. Serial Mode Connections to the External Transceiver

Signal Description	FEC Signal Name
Transmit clock	TX_CLK
Transmit enable	TX_EN
Transmit data	TXD0
Collision	COL
Receive clock	RX_CLK
Receive enable	RX_DV
Receive data	RXD0
Unused MPC855T inputs—Tie to ground	RX_ER, CRS, RXD[3:1]
Unused MPC855T outputs—Ignore	TX_ER, TXD[3:1], MDC, MDIO

43.2.2 FEC Frame Transmission

FEC transmissions require almost no host intervention. When the software driver sets the ETHER_EN bit in the Ethernet control register (ECNTRL) and the X_DES_ACTIVE bit in the CSR TxBD active register (X_DES_ACTIVE), the FEC is enabled and fetches the first TxBD. If the user has a frame ready to transmit, a DMA transfer of the transmit data buffers begins immediately.

A 512-bit collision window of transmit data is sent to the transmit FIFO before transmission begins. If the line is not busy, the MAC transmit logic asserts TX_EN and sends the preamble sequence, the start frame delimiter (SFD), and then the frame information. If the line is busy, the controller waits for the carrier sense signal, CRS, to remain inactive for 60 bit times. Transmission begins after an additional 36 bit times (96 bit times after CRS became inactive).

If a collision occurs during the transmit frame, the FEC follows the specified backoff procedures and tries retransmitting the frame until the retry limit threshold is reached. The FEC stores the first 64 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 usage and latency in case the backoff timer output causes a need for an immediate retransmission.

When the end of the current BD is reached and TxBD[L] is set, the frame check sequence (32-bit CRC) is appended (if TxBD[TC] = 1) and TX_EN is negated. After the frame check sequence is sent, the FEC writes the frame status bits into the BD and clears the R bit. When the end of the current BD is reached and the L bit is not set (a frame consists of multiple buffers), only the R bit is cleared. Short frames are automatically padded by the transmit logic.

A transmit frame length exceeding the value set for MAX_FRAME_LENGTH in the receive hash register (R_HASH) generates a babbling transmit interrupt (I_EVENT[BABT] = 1); however, the entire frame is sent (no truncation). Whether buffer or frame interrupts can be generated is determined by I_MASK settings.

To pause transmission, set the graceful transmit stop bit, X_CNTRL[GTS]. When GTS is set, the FEC transmitter stops immediately if no transmission is in progress or continues transmission until the current frame either finishes or terminates with a collision. The GRA interrupt occurs when the graceful transmit stop operation completes. When GTS is cleared, the FEC resumes transmission with the next frame.

The FEC transmits bytes lsb first.

43.2.3 FEC Frame Reception

FEC reception requires almost no host intervention. The FEC can perform address recognition, CRC checking, short-frame checking, and maximum frame-length checking.

When the software driver sets ECNTRL[ETHER_EN] and R_DES_ACTIVE in the CSR RxBD active register (R_DES_ACTIVE), the FEC receiver is enabled and immediately starts processing receive frames. When RX_DV is asserted, the receiver first checks for a valid preamble/SFD (start frame delimiter) header, which is stripped and the frame is processed by the receiver. If a valid header is not found, the frame is ignored.

When in serial mode and after RX_DV (RENA) is asserted, the first 16 bit times of RX_D0 are ignored. Following the first 16 bit times the data sequence is checked for alternating ones and zeros.

- If a 11 or 00 sequence is detected during bit times 17 to 21, the rest of the frame is ignored.
- After bit time 21, the data sequence is monitored for a valid SFD (11). If a 00 is detected, the frame is rejected. If a 11 is detected, the preamble/SFD sequence is complete.

In MII mode, the receiver checks for at least one byte matching the SFD. Zero or more preamble bytes may occur, but if a 00 sequence is detected before the SFD byte, the frame is ignored.

After the first eight bytes of the frame are passed to the receive FIFO, the FEC performs address recognition on the frame.

As soon as a collision window (64 bytes) of data is received and if address recognition has not rejected the frame, the FEC starts transferring the incoming frame to the RxBD's associated buffer. If the frame is a too short (due to collision) or is rejected by address recognition, no receive buffers are filled. Thus, no collision frames are presented to the user, except for any late collisions, which indicate serious LAN problems. When the data buffer has been filled, the FEC clears RxBD[E] and generates an RXB interrupt (if I_MASK[RBIEN] is set). If the incoming frame exceeds the length of the data buffer, the FEC fetches the next RxBD in the table. If it is empty, the FEC continues transferring the rest of the frame to the associated data buffer.

R_BUFF_SIZE[R_BUFF_SIZE] determines buffer length, which should be at least 128 bytes. R_BUFF_SIZE must be quad-word (16-byte) aligned.

During reception, the FEC checks for a frame that is either too short or too long. When the frame ends (CRS is negated), the receive CRC field is checked and written to the data buffer. The data length written to the last BD in the Ethernet frame is the length of the entire frame. Frames smaller than 64 bytes are not accessed and are rejected in hardware with no impact on system bus usage.

Receive frames are not truncated if they exceed MAX_FRAME_LENGTH bytes, however the babbling receive error interrupt occurs (I_EVENT[BABR] = 1) and RxBD[LG] is set.

When the receive frame is complete, the FEC sets RxBD[L], writes the other frame status bits into the RxBD, and clears the E bit. The FEC next generates a maskable interrupt (I_EVENT[RFINT] maskable by I_MASK[RFIEN]), indicating that a frame has been received and is in memory. The FEC then waits for a new frame.

The FEC receives serial data lsb first.

43.2.4 CAM Interface

In addition to the FEC address recognition logic, an external CAM may be used for frame reject with no additional pins other than the MII interface pins. For more information on the CAM interface refer to *Using Motorola's Fast Static RAM CAMs with the MPC860T's Media Independent Interface* application note.

43.2.5 FEC Command Set

The FEC does not support commands as found in the CPM channels. After the FEC is initialized and enabled, it operates autonomously. Typically, aside from initialization, the driver only writes to R_DES_ACTIVE, X_DES_ACTIVE, and I_EVENT during operation.

43.2.6 Ethernet Address Recognition

The FEC filters the received frames based on destination address (DA) type—individual (unicast), group (multicast), or broadcast (all-ones group address). The difference between an individual address and a group address is determined by the I/G bit in the destination address field.

If the DA is the individual (unicast) type of address, the FEC compares the destination address field of the received frame with the 48-bit address that the user programs in the ADDR_LOW and ADDR_HIGH.

If the DA is the group type of address, the FEC determines whether the group address is a broadcast address. If it is, the frame is accepted unconditionally; otherwise (multicast address) a hash table lookup is performed using the 64-entry hash table defined in the hash table registers.

In promiscuous mode (R_CNTRL[PROM] = 1), the FEC receives all the incoming frames regardless of their address. In this mode the DA lookup is still performed and the MISS bit in the RxBD is set accordingly. If address recognition did not achieve a match, the frame is received with RxBD[MISS] set. If address recognition achieves a match, the frame is received without the MISS bit being set.

Figure 43-1 shows a flowchart for address recognition on received frames.

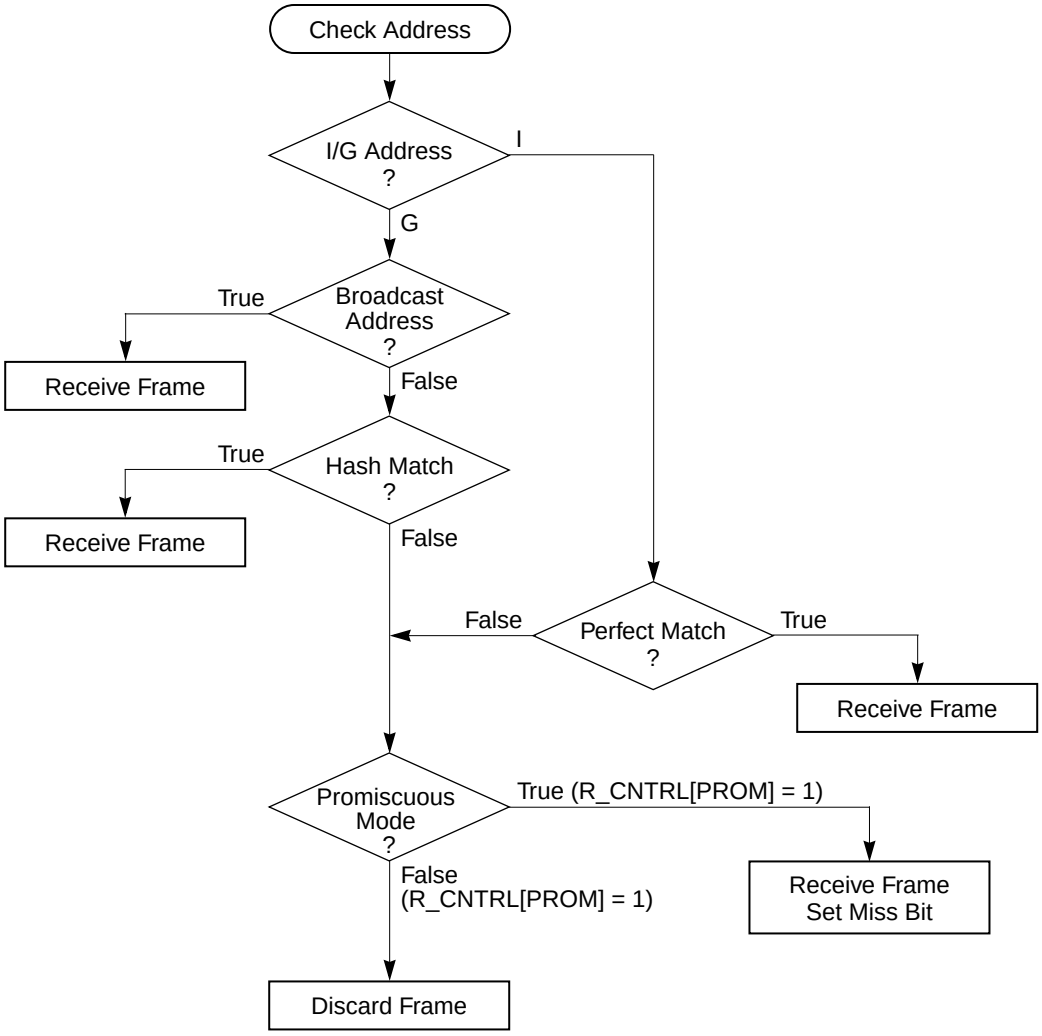


Figure 43-2. Ethernet Address Recognition Flowchart

43.2.7 Hash Table Algorithm

This section discusses the hash table process used in group hash filtering. When the FEC receives a frame with the destination address I/G bit set, the 48-bit address is mapped into one of 64 bins, represented by the 64 bits in the two hash table registers. This is performed 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 0 and 63.

Bit 31 of the CRC result selects HASH_TABLE_HIGH (bit 31 = 1) or HASH_TABLE_LOW (bit 31 = 0). Bits 30–26 of the CRC result select the bit in the selected register. If that bit is set in the 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 (or 87.5%) of the group

address frames from reaching memory. The processor must further filter those that reach memory to determine if they truly contain one of the eight preferred addresses.

The effectiveness of the hash table declines as the number of addresses increases.

The user must initialize the hash table registers. The FEC does not support the SET GROUP ADDRESS command, which can be used in CPM ethernet controllers. The user may compute the hash for a particular address in software or use the SET GROUP ADDRESS command in an off-line CPM channel, retrieve the result, and use it to program the FEC hash table registers. The CRC32 polynomial to use in computing the hash is as follows:

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

43.2.8 Inter-Packet Gap Time

The minimum inter-packet gap time for back-to-back transmission is 96 bit times. After completing a transmission or after the backoff algorithm completes, the transmitter waits for the carrier sense signal (CRS) to be negated before starting its 96 bit time IPG counter. Frame transmission may begin 96 bit times after CRS is negated if it stays negated for at least 60 bit times. If CRS asserts during the last 36 bit times, it is ignored and a collision occurs.

The receiver receives back-to-back frames with a minimum spacing of at least 28 bit times. If an interrupted gap between receive frames is less than 28 bit times, the receiver may discard the next frame.

43.2.9 Collision Handling

If a collision occurs during frame transmission, the FEC continues transmitting for at least 32 bit times, sending a JAM pattern of 32 ones. If the collision occurs during the preamble sequence, the JAM pattern is sent after 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. A slot time is 512 bit times. If a collision occurs after 64 byte times, no retransmission is performed and the end of frame buffer is closed with an LC error indication.

43.2.10 Internal and External Loopback

The FEC supports both internal and external loopback. In loopback mode, both FIFOs are used and the FEC operates in full-duplex fashion. Both internal and external loopback are configured through R_CNTRL[LOOP, DRT].

For internal loopback, set LOOP = 1 and DRT = 0. TX_EN and TX_ER are not asserted during internal loopback.

For external loopback, set LOOP = 0 and DRT = 0. Configure the external transceiver for loopback.

43.2.11 Ethernet Error-Handling Procedure

The FEC reports frame reception and transmission error conditions using the FEC BDs and the I_EVENT register.

43.2.11.1 Transmission Errors

Table 43-3 describes transmission errors.

Table 43-3. Transmission Errors

Error	Description
Transmitter Underrun	If this error occurs, the FEC sends 32 bits that ensure a CRC error and stops transmitting. All remaining buffers for that frame are then flushed and closed, with the UN bit set in the last TxBD for that frame. The FEC continues to the next TxBD and begins transmitting the next frame.
Carrier Sense Lost during Frame Transmission	When this error occurs and no collision is detected in the frame, the FEC sets the CSL bit in the last TxBD for this frame. The frame is sent normally. No retries are performed as a result of this error. The CSL bit is not set if X_CNTRL[FDEN] = 1, regardless of the state of CRS.
Retransmission Attempts Limit Expired	When this error occurs, the FEC terminates transmission. All remaining buffers for that frame are then flushed and closed, with the RL bit set in the last TxBD for that frame. The FEC then continues to the next TxBD and begins sending the next frame.
Late Collision	When this error occurs, the FEC stops sending. All remaining buffers for that frame are then flushed and closed, with the LC bit set in the last TxBD for that frame. The FEC then continues to the next TxBD and begins sending the next frame. Note: The definition of what constitutes a late collision is hard-wired in the FEC.
Heartbeat	Some transceivers have a self-test feature called heartbeat or signal-quality error. To signify a good self-test, the transceiver indicates a collision within 20 clocks after the FEC sends a frame. This heartbeat condition does not imply a real collision, but that the transceiver seems to work properly. If X_CNTRL[HBC] = 1, X_CNTRL[FDEN]=0, and a heartbeat condition is not detected after a frame transmission, a heartbeat error occurs—the FEC closes the buffer, sets TxBD[HB], and generates the HBERR interrupt if it is enabled.

43.2.11.2 Reception Errors

Table 43-4 describes reception errors.

Table 43-4. Reception Errors

Error	Description
Overrun Error	The FEC maintains an internal FIFO for receiving data. If a receiver FIFO overrun occurs, the FEC closes the buffer and sets RxBD[OV].
Non-Octet Error (Dribbling Bits)	The FEC 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, the frame nonoctet aligned (NO) error is reported in the RxBD. If there is no CRC error, no error is reported.

Table 43-4. Reception Errors

Error	Description
CRC Error	When a CRC error occurs with no dribbling bits, the FEC closes the buffer and sets RxBD[CR]. CRC checking cannot be disabled, but the CRC error can be ignored if checking is not required.
Frame Length Violation	When the receive frame length exceeds R_HASH[MAX_FRAME_LENGTH], I_EVENT[BABR] is set indicating babbling receive error, and the LG bit in the end of frame RxBD is set. Note: Receive frames exceeding 2047 bytes are truncated.

43.2.12 SDMA Bus Arbitration and Transfers

There are two arbitration levels to consider—accesses to the SDMA hardware and accesses to the 60x bus. As shown in Figure 43-3, if the CPM and the 100BASE-T module attempt to access the SDMA simultaneously, the CPM wins the first access. If both continue to request the SDMA hardware, control alternates between the two.

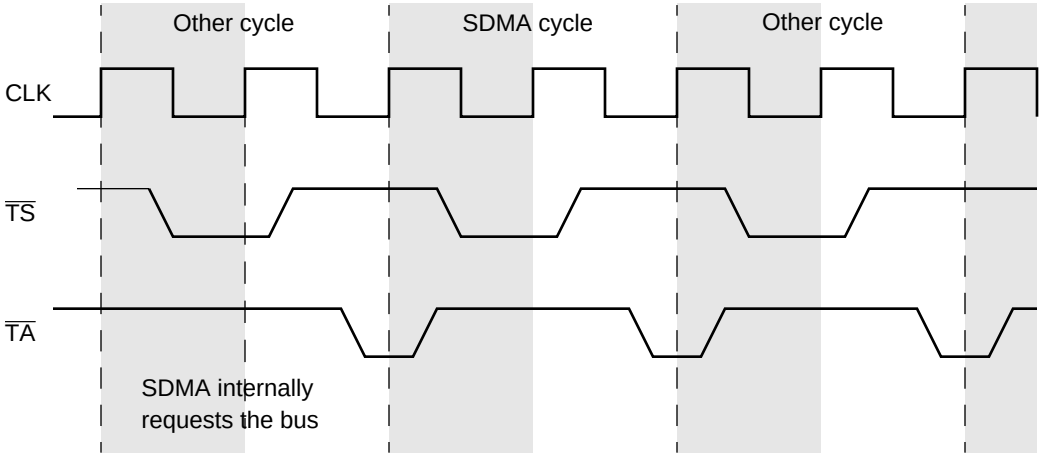


Figure 43-3. SDMA Bus Arbitration

The priority of the SDMA on the 60x bus is programmed in SDCR[RAID], described in Section 43.2.13.1, “SDMA Configuration Register (SDCR).”

43.2.13 The SDMA Registers

The SDMA channels share a configuration register, address register, and status register, and are controlled by the configuration of the SCC, SMCs, SPI, and I^2C controllers.

43.2.13.1 SDMA Configuration Register (SDCR)

The SDMA configuration register (SDCR) interacts with the DMA controllers in the FEC, see Section 19.2.1, “SDMA Configuration Register (SDCR).

43.3 Signal Descriptions

TheMPC855T system bus signals consist of all the lines that interface with the external bus. Many of these lines perform different functions, depending on how the user assigns them. The input and output signals, shown in Table 43-5, are identified by their abbreviated names. There may be other multiplexed functionality on some of these pins. Refer to Chapter 12, “External Signals.

Table 43-5. FEC Signal Descriptions

Name	Pin Number	Description
$\overline{\text{IRQ7}}$ MII_TX_CLK	W15	Interrupt request 7—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. MII transmit clock—Input clock that provides the timing reference for TX_EN, TXD, and TX_ER. Note that MII_TXCLK becomes active as soon as ECNTRL[ETHER_EN] is set. $\overline{\text{IRQ7}}$ must be masked in the SIU, see Section 10.5.4.2, “SIU Interrupt Mask Register (SIMASK).
PD[15] L1TSYNCA MII_RXD[3]	U17	General-purpose I/O port D bit 15—This is bit 15 of the general-purpose I/O port D. Transmit data sync signal for TDM channel A MII receive data 3—Input signal RXD[3] represents bit 3 of the nibble of data to be transferred from the PHY to the MAC when RX_DV is asserted.
PD[14] L1RSYNCA MII_RXD[2]	V19	General-purpose I/O port D bit 14—This is bit 14 of the general-purpose I/O port D. Input receive data sync signal to the TDM channel A MII receive data 2—Input signal RXD[2] represents bit 2 of the nibble of data to be transferred from the PHY to the MAC when RX_DV is asserted.
PD[13] MII_RXD[1]	V18	General-purpose I/O port D bit 13—This is bit 13 of the general-purpose I/O port D. MII receive data 1—Input signal RXD[1] represents bit 1 of the nibble of data to be transferred from the PHY to the MAC when RX_DV is asserted.
PD[12] MII_MDC	R16	General-purpose I/O port D bit 12—This is bit 12 of the general-purpose I/O port D. MII management data clock—Output clock provides a timing reference to the PHY for data transfers on the MDIO signal.
PD[11] MII_TX_ER	T16	General-purpose I/O port D bit 11—This is bit 11 of the general-purpose I/O port D. MII transmit error—Output signal when asserted for one or more clock cycles while TX_EN is asserted shall cause the PHY to transmit one or more illegal symbols. Asserting TX_ER has no effect when operating at 10 Mbps or when TX_EN is negated.
PD[10] MII_RXD[0]	W18	General-purpose I/O port D bit 10—This is bit 10 of the general-purpose I/O port D. MII receive data 0—Input signal RXD[0] represents bit 0 of the nibble of data to be transferred from the PHY to the MAC when RX_DV is asserted. In 10 Mbps serial mode, RXD[0] is used and RXD[1–3] are ignored.
PD[9] MII_TXD[0]	V17	General-purpose I/O port D bit 9—This is bit 9 of the general-purpose I/O port D. MII transmit data 0—Output signal TXD[0] represents bit 0 of the nibble of data when TX_EN is asserted and has no meaning when TX_EN is negated. In 10Mbps serial mode, TXD[0] is used and TXD[1–3] are ignored.

Table 43-5. FEC Signal Descriptions (continued)

Name	Pin Number	Description
PD[8] MII_RX_CLK	W17	General-purpose I/O port D bit 8—This is bit 8 of the general-purpose I/O port D. MII receive clock—Input clock which provides a timing reference for RX_DV, RXD, and RX_ER.
PD[7] MII_RX_ER	T15	General-purpose I/O port D bit 7—This is bit 7 of the general-purpose I/O port D. MII receive error—When Input signal RX_ER and RX_DV are asserted, the PHY has detected an error in the current frame. When RX_DV is not asserted, RX_ER has no effect.
PD[6] MII_RX_DV	V16	General-purpose I/O port D bit 6—This is bit 6 of the general-purpose I/O port D. MII receive data valid—When input signal RX_DV is asserted, the PHY is indicating that a valid nibble is present on the MII. This signal shall remain asserted from the first recovered nibble of the frame through the last nibble. Assertion of RX_DV must start no later than the SFD and exclude any EOF.
PD[5] MII_TXD[3]	U15	General-purpose I/O port D bit 5—This is bit 5 of the general-purpose I/O port D. MII transmit data 3—Output signal TXD[3] represents bit 3 of the nibble of data when TX_EN is asserted and has no meaning when TX_EN is negated.
PD[4] MII_TXD[2]	U16	General-purpose I/O port D bit 4—This is bit 4 of the general-purpose I/O port D. MII transmit data 2—Output signal TXD[2] represents bit 2 of the nibble of data when TX_EN is asserted and has no meaning when TX_EN is negated.
PD[3] MII_TXD[1]	W16	General-purpose I/O port D bit 3—This is bit 3 of the general-purpose I/O port D. MII transmit data 1—Output signal TXD[1] represents bit 1 of the nibble of data when TX_EN is asserted and has no meaning when TX_EN is negated.
MII_TX_EN	V15	MII transmit enable—Output signal TX_EN indicates when there are valid nibbles being presented on the MII. This signal is asserted with the first nibble of preamble and is negated prior to the first TX_CLK following the final nibble of the frame. This signal resets to three-state with a weak internal pull-down to ensure compatibility with 860 applications that may have tied SPARE3 (V15) to VCC or GND. This pin will be 3-V only and must not be pulled up to +5 V.
MII_CRS	B7	MII carrier receive sense—When input signal CRS is asserted the transmit or receive medium is not idle. In the event of a collision, CRS will remain asserted through the duration of the collision.
MII_COL	H4	MII collision—Input signal COL is asserted upon detection of a collision, and will remain asserted while the collision persists. The behavior of this signal is not specified for full-duplex mode.
MII_MDIO	H18	MII management data—Bidirectional signal, MDIO transfers control information between the PHY and MAC. Transitions synchronously to MDC.

43.4 Programming Model

The FEC software model is similar to that used by the 10-Mbps Ethernet implemented on the CPM. To support higher data rates, the FEC has a different internal architecture, which changes the programming model slightly. However, efforts have been taken to minimize the differences required by the interrupt handlers. The FEC's registers are very different from those of the CPM-based internal Ethernet controller.

The FEC is programmed by a combination of control/status registers (CSRs) and BDs. The CSRs are used for mode control and to extract global status information. The BDs are used to pass data buffers and related buffer information between hardware and software.

Some registers are located in on-chip RAM. All on-chip registers, whether located in RAM or in hardware, must be accessed using big-endian mode. Therefore, descriptions in this chapter assume big-endian byte ordering. There is no support for little-endian in the FEC.

43.4.1 Parameter RAM

Table 43-6 describes each entry in the FEC parameter RAM.

Table 43-6. FEC Parameter RAM Memory Map

Address	Name	Description	Section
0xE00	ADDR_LOW	Lower 32 bits of address	Section 43.4.1.1, "RAM Perfect Match Address Low Register (ADDR_LOW)"
0xE04	ADDR_HIGH	Upper 16 bits of address	Section 43.4.1.2, "RAM Perfect Match Address High (ADDR_HIGH)"
0xE08	HASH_TABLE_HIGH	Upper 32 bits of hash table	Section 43.4.1.3, "RAM Hash Table High (HASH_TABLE_HIGH)
0xE0C	HASH_TABLE_LOW	Lower 32 bits of hash table	Section 43.4.1.4, "RAM Hash Table Low (HASH_TABLE_LOW)
0xE10	R_DES_START	Pointer to beginning of RxBD ring	Section 43.4.1.5, "Beginning of RxBD Ring (R_DES_START)
0xE14	X_DES_START	Pointer to beginning of TxBD ring	Section 43.4.1.6, "Beginning of TxBD Ring (X_DES_START)
0xE18	R_BUFF_SIZE	Receive buffer size	Section 43.4.1.7, "Receive Buffer Size Register (R_BUFF_SIZE)
0xE40	ECNTRL	Ethernet control register	Section 43.4.1.8, "Ethernet Control Register (ECNTRL)
0xE44	IEVENT	Interrupt event register	Section 43.4.1.9, "Interrupt Event (I_EVENT)/Interrupt Mask Register (I_MASK)

Table 43-6. FEC Parameter RAM Memory Map (continued)

Address	Name	Description	Section
0xE48	IMASK	Interrupt mask register	Section 43.4.1.9, “Interrupt Event (I_EVENT)/Interrupt Mask Register (I_MASK)
0xE4C	IVEC	Interrupt level and vector status	Section 43.4.1.10, “Ethernet Interrupt Vector Register (IVEC)
0xE50	R_DES_ACTIVE	Receive ring updated flag	Section 43.4.1.11, “RxBD Active Register (R_DES_ACTIVE)
0xE54	X_DES_ACTIVE	Transmit ring updated flag	Section 43.4.1.12, “TxBD Active Register (X_DES_ACTIVE)
0xE80	MII_DATA	MII data register	Section , “
0xE84	MII_SPEED	MII speed register	Section 43.4.1.14, “MII Speed Control Register (MII_SPEED)
0xECC	R_BOUND	End of FIFO RAM (read-only)	Section 43.4.1.15, “FIFO Receive Bound Register (R_BOUND)
0xED0	R_FSTART	Receive FIFO start address	Section 43.4.1.16, “FIFO Receive Start Register (R_FSTART)
0xEE4	X_WMRK	Transmit watermark	Section 43.4.1.17, “Transmit Watermark Register (X_WMRK)
0xEEC	X_FSTART	Transmit FIFO start address	Section 43.4.1.18, “FIFO Transmit Start Register (X_FSTART)
0xF34	FUN_CODE	Function code to SDMA	Section 43.4.1.19, “DMA Function Code Register (FUN_CODE)
0xF44	R_CNTRL	Receive control register	Section 43.4.1.20, “Receive Control Register (R_CNTRL)
0xF48	R_HASH	Receive hash register	Section 43.4.1.21, “Receive Hash Register (R_HASH)
0xF84	X_CNTRL	Transmit control register	Section 43.4.1.22, “Transmit Control Register (X_CNTRL)

43.4.1.1 RAM Perfect Match Address Low Register (ADDR_LOW)

The ADDR_LOW register, shown in Figure 43-4, is written by and must be initialized by the user. It contains the lower 32 bits of the 48-bit address used in the address recognition process to compare with the destination address field of the receive frames.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	ADDR_LOW BYTE 0								ADDR_LOW BYTE 1							
Reset	Undefined															
R/W	Read/write															
Addr	0xE00															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	ADDR_LOW BYTE 2								ADDR_LOW BYTE 3							
Reset	Undefined															
R/W	Read/write															
Addr	0xE02															

Figure 43-4. ADDR_LOW Register

Table 43-7 describes the ADDR_LOW fields.

Table 43-7. ADDR_LOW Field Descriptions

Bits	Name	Description
0–31	ADDR_LOW	Bytes in the 6-byte address: 0 (bits 0–7), 1 (bits 8–15), 2 (bits 16–23) and 3 (bits 24–31)

43.4.1.2 RAM Perfect Match Address High (ADDR_HIGH)

The ADDR_HIGH register, shown in Figure 43-5, is written by and must be initialized by the user. It contains bytes 4 and 5 of the 6-byte address used to compare with the destination address field of the receive frames. Byte 0 is the first byte sent at the start of the frame.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	ADDR_HIGH BYTE 4								ADDR_HIGH BYTE 5							
Reset	Undefined															
R/W	Read/write															
Addr	0xE04															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—															
Reset	Undefined															
R/W	Read/write															
Addr	0xE06															

Figure 43-5. ADDR_HIGH Register

Table 43-8 describes the ADDR_HIGH fields.

Table 43-8. ADDR_HIGH Field Descriptions

Bits	Name	Description
0–15	ADDR_HIGH	Bytes of the 6-byte address: 4 (bits 0–7) and 5 (bits 8–15)
16–31	—	Reserved. Should be cleared.

43.4.1.3 RAM Hash Table High (HASH_TABLE_HIGH)

The HASH_TABLE_HIGH register, shown in Figure 43-6, contains the upper 32 bits of the 64-bit hash table used in address recognition for receive frames with a multicast address. It is written by and must be initialized by the user

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	HASH_HIGH															
Reset	Undefined															
R/W	Read/write															
Addr	0xE08															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	HASH_HIGH															
Reset	Undefined															
R/W	Read/write															
Addr	0xE0A															

Figure 43-6. HASH_TABLE_HIGH Register

Table 43-9 describes HASH_TABLE_HIGH fields.

Table 43-9. HASH_TABLE_HIGH Field Descriptions

Bits	Name	Description
0–31	HASH_HIGH	Contains the upper 32 bits of the 64-bit hash table used in address recognition for receive frames with a multicast address. HASH_HIGH[0] contains hash index bit 63. HASH_HIGH[31] contains hash index bit 32.

43.4.1.4 RAM Hash Table Low (HASH_TABLE_LOW)

The HASH_TABLE_LOW register, shown in Figure 43-7, contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. It is written by and must be initialized by the user.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	HASH_LOW															
Reset	Undefined															
R/W	Read/write															
Addr	0xE0C															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	HASH_LOW															
Reset	Undefined															
R/W	Read/write															
Addr	0xE0E															

Figure 43-7. HASH_TABLE_LOW Register

Table 43-10 describes HASH_TABLE_LOW fields.

Table 43-10. HASH_TABLE_LOW Field Descriptions

Bits	Name	Description
0–31	HASH_LOW	Contains the lower 32 bits of the 64-bit hash table used in address recognition for receive frames with a multicast address. HASH_LOW[0] contains hash index bit 31. HASH_LOW[31] contains hash index bit 0.

43.4.1.5 Beginning of RxBD Ring (R_DES_START)

The R_DES_START register, shown in Figure 43-8, is like the RBASE register used by other protocols. It provides a pointer to the start of the circular RxBD queue in external memory.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	R_DES_START															
Reset	Undefined															
R/W	Read/write															
Addr	0xE10															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	R_DES_START														00	
Reset	Undefined															
R/W	Read/write															
Addr	0xE12															

Figure 43-8. R_DES_START Register

Table 43-11 describes R_DES_START fields.

Table 43-11. R_DES_START Field Descriptions

Bits	Name	Description
0–29	R_DES_START	Pointer to start of RxBD queue.
30–31	—	Reserved. Should be written to zero by the host processor.

43.4.1.6 Beginning of TxBD Ring (X_DES_START)

The X_DES_START register, shown in Figure 43-9, is like the TBASE register used by other protocols. It provides a pointer to the start of the circular TxBD queue in external memory.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	X_DES_START															
Reset	Undefined															
R/W	Read/write															
Addr	0xE14															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	X_DES_START														00	
Reset	Undefined															
R/W	Read/write															
Addr	0xE16															

Figure 43-9. X_DES_START Register

Table 43-12 describes X_DES_START fields.

Table 43-12. X_DES_START Field Descriptions

Bits	Name	Description
0–29	X_DES_START	Pointer to start of TxBD queue.
30–31	—	Reserved. Should be written to zero by the host processor.

43.4.1.7 Receive Buffer Size Register (R_BUFF_SIZE)

The R_BUFF_SIZE register, shown in Figure 43-10, is like the MRBLR register used by other protocols. It specifies the maximum size of all receive buffers. It does not reset and must be initialized by the user. Because the maximum frame is 2047 bytes, only bits 21–27 are used. The user should take into consideration that the receive CRC is always written into the last receive buffer. To support frame lengths up to 1520 bytes, R_BUFF_SIZE must be at least 0x0000_05F0. To ensure that R_BUFF_SIZE is a multiple of 16, bits 28–31 are forced to zeros. Using buffers smaller than the recommended minimum 256 bytes increases the risk of receive FIFO overflow due to the overhead of opening and closing buffers.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Reset	Undefined															
R/W	Read/write															
Addr	0xE18															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—					R_BUFF_SIZE							—			
Reset	Undefined															
R/W	Read/write															
Addr	0xE1A															

Figure 43-10. R_BUFF_SIZE Register

Table 43-13 describes R_BUFF_SIZE fields.

Table 43-13. R_BUFF_SIZE Field Descriptions

Bits	Name	Description
0–20	—	Reserved. Should be written to zero by the host processor.
21–27	R_BUFF_SIZE	Receive buffer size.
28–31	—	Reserved. Should be written to zero by the host processor.

43.4.1.8 Ethernet Control Register (ECNTRL)

The Ethernet control register (ECNTRL), shown in Figure 43-11, is used to enable and disable the FEC. It is written by the user and cleared at system reset.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xE40															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	SPARE													FEC_PIN MUX	ETHER_EN	RESET
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xE42															

Figure 43-11. ECNTRL Register

Table 43-14 describes ECNTRL fields.

Table 43-14. ECNTRL Field Descriptions

Bits	Name	Description
0–28	—	Reserved. These fields may return unpredictable values and should be masked on a read. Users should always write these fields to zero.
29	FEC_PINMUX	FEC enable. Read/write. The user must set this bit to enable the FEC function in the 860 in conjunction with 860 pin multiplexing control.
30	ETHER_EN	Ethernet enable. 0 A transfer is stopped after a bad CRC is appended to any frame being sent. 1 The FEC is enabled, and reception and transmission are possible. The BDs for an aborted transmit frame are not updated after ETHER_EN is cleared. When ETHER_EN is cleared, the DMA, BD, and FIFO control logic are reset including BD and FIFO pointers. See Section 43.4.2.2, “User Initialization (before Setting ECNTRL[ETHER_EN]).”
31	RESET	Ethernet controller reset. When RESET = 1, the equivalent of a hardware or software reset is performed but it is local to the FEC. ETHER_EN is cleared and all other FEC registers take their reset values. Also, any transfers are abruptly aborted. Hardware automatically clears RESET once the hardware reset is complete (approximately 16 clock cycles).

43.4.1.9 Interrupt Event (I_EVENT)/Interrupt Mask Register (I_MASK)

When an event sets a bit in the interrupt event register (I_EVENT), shown in Figure 43-12, an interrupt is generated if the corresponding interrupt mask register (I_MASK) bit is set. I_EVENT bits are cleared by writing ones; writing zeros has no effect.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	HBERR	BABR	BABT	GRA	TFINT	TXB	RFINT	RXB	MII	EBERR	—					
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xE44 (I_EVENT); 0xE48 (I_MASK)															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—															
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xE46(I_EVENT); 0xE4A (I_MASK)															

Figure 43-12. I_EVENT/I_MASK Registers

Table 43-15 describes I_EVENT and I_MASK fields. Note that neither the RxBD or TxBD has an I bit to enable/disable an interrupt on the receive or transmit buffer. As events occur, they are always reported in I_EVENT, but only those not masked in I_MASK cause an interrupt. From a system resources and software performance standpoint, it is advisable to minimize the number of interrupts per frame by masking TXB and RXB in favor of TFINT and RFINT to notify at the end of frame.

Table 43-15. I_EVENT/I_MASK Field Descriptions

Bits	Name	Description
0	HBERR	Heartbeat error. When I_EVENT[HBC] is set, this interrupt indicates that heartbeat was not detected within the heartbeat window following a transmission.
1	BABR	Babbling receive error. Indicates a received frame exceeded MAX_FRAME_LENGTH bytes. The hardware truncates receive frames exceeding 2047 bytes so as not to overflow receive buffers.
2	BABT	Babbling transmit error. Indicates that the transmitted frame exceeded MAX_FRAME_LENGTH bytes. This condition is usually caused by too large a a frame being placed into the transmit data buffers. The transmit frame is not truncated.
3	GRA	Graceful stop complete. A graceful stop initiated by the setting of GTS is complete. GRA is set when the transmitter finishes sending any frame that was in progress when GTS was set.
4	TFINT	Transmit frame interrupt. Indicates that a frame was sent and that the last corresponding BD was updated.
5	TXB	Transmit buffer interrupt. Indicates that a TxBD was updated.
6	RFINT	Receive frame interrupt. Indicates that a frame was received and that the last corresponding BD was updated.
7	RXB	Receive buffer interrupt. Indicates that a RxBD was updated.
8	MII	MII interrupt. Indicates that the MII completed the requested data transfer.
9	EBERR	Ethernet bus error occurred. Indicates that a bus error occurred when the FEC was accessing the U bus.
10–31	—	Reserved. Should written to zero by the host processor.

43.4.1.10 Ethernet Interrupt Vector Register (IVEC)

The Ethernet interrupt vector register (IVEC), shown in Table 43-16, indicates the class of interrupt generated by the FEC (IVEC) and provides control of the interrupt level (ILEVEL).

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	ILEVEL			—												
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xE4C															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—												IVEC		—	
Reset	0000_0000_0000_0000															
R/W	—												Read only		—	
Addr	0xE4E															

Figure 43-13. IVEC Register

Table 43-16 describes IVEC fields.

Table 43-16. IVEC Field Descriptions

Bits	Name	Description
0–2	ILEVEL	Interrupt level. The ILEVEL is used to define the interrupt level (0–7) associated with the FEC interrupt (one of the SIU internal interrupt sources).
3	—	Reserved. Should be written to zero by the host processor.
4–5	—	Reserved. Should be written to zero by the host processor.This field may return unpredictable values and should be masked on a read
6–27	—	Reserved. Should be written to zero by the host processor.
28–29	IVEC	Interrupt vector, read only. IVEC gives the highest outstanding priority Fast Ethernet interrupt. The bit field meanings (from low priority to high priority) are as follows: 00 No pending FEC interrupt 01 Non-time-critical interrupt 10 Transmit interrupt 11 Receive interrupt
30–31	—	Reserved. Should be written to zero by the host processor.

43.4.1.11 RxBD Active Register (R_DES_ACTIVE)

The RxBD active register (R_DES_ACTIVE), shown in Figure 43-14, is a command register that should be written by the user to indicate that the RxBD ring was updated (empty receive buffers have been produced by the software driver with the E bit set).

Whenever the register is written, the R_DES_ACTIVE bit is set, regardless of the data written by the user. While the bit is set, the RxBD ring is polled and receive frames (provided ECNTRL[ETHER_EN] is also set) are processed. Once an RxBD whose ownership bit is not set is polled, the R_DES_ACTIVE bit is cleared and polling stops until the user sets the bit again, signifying additional BDs have been placed into the RxBD ring.

R_DES_ACTIVE is cleared at reset and by clearing ECNTRL[ETHER_EN].

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—							R_DES_ACTIVE	—							
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xE50															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xE52															

Figure 43-14. R_DES_ACTIVE Register

Table 43-17 describes R_DES_ACTIVE fields.

Table 43-17. R_DES_ACTIVE Field Descriptions

Bits	Name	Description
0–6	—	Reserved.
7	R_DES_ACTIVE	Signals the FEC that empty buffers are available. Set when this register is written, regardless of the value written. Cleared by the FEC whenever no additional BDs are ready in the RxBD ring.
8–31	—	Reserved.

43.4.1.12 TxBD Active Register (X_DES_ACTIVE)

The TxBD active register, shown in Figure 43-15, is a command register that the user should write to indicate that the TxBD ring was updated (transmit buffers have been produced by the software driver with TxBD[R] set).

Whenever the register is written, X_DES_ACTIVE is set, regardless of the data written by the user. When the bit is set, the TxBD ring is polled and transmit frames (provided ECNTRL[ETHER_EN] is also set) are processed. Once a TxBD whose ownership bit is not set is polled, X_DES_ACTIVE is cleared and polling stops until the bit is set, signifying additional BDs have been placed into the TxBD ring.

X_DES_ACTIVE is cleared at reset and by clearing ECNTRL[ETHER_EN].

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—							X_DES_ACTIVE	—							
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xE54															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—															
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xE56															

Figure 43-15. X_DES_ACTIVE Register

43.4.1.13 MII Management Frame Register (MII_DATA)

Table 43-18 describes X_DES_ACTIVE fields.

Table 43-18. X_DES_ACTIVE Field Descriptions

Bits	Name	Description
0–6	—	Reserved.
7	X_DES_ACTIVE	Set when this register is written, regardless of the value written. Cleared whenever no additional ready descriptors remain in the transmit ring.
8–31	—	Reserved.

The MII_DATA register, shown in Figure 43-16, is used to communicate with the attached MII-compatible PHY device, providing read/write access to their MII registers. Writing to MII_DATA causes a management frame to be sourced unless MII_SPEED was cleared; in this case, if MII_SPEED is then written to a non-zero value and an MII frame is generated with the data previously written to MII_DATA. This allows MII_DATA and MII_SPEED to be programmed in either order if MII_SPEED is currently zero. MII_DATA is accessed by the user and does not reset to a defined value.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	ST		OP		PA					RA					TA	
Reset	Undefined															
R/W	Read/write															
Addr	0xE80															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	DATA															
Reset	Undefined															
R/W	Read/write															
Addr	0xE82															

Figure 43-16. MII_DATA Register

Table 43-19 describes MII_DATA fields.

Table 43-19. MII_DATA Field Descriptions

Bits	Name	Description
0–1	ST	Start of frame delimiter. Must be programmed to 01 for a valid MII management frame.
2–3	OP	Operation code. Must be 10 (read) or 01(write) to generate a valid MII management frame.
4–8	PA	PHY address. Specifies one of up to 32 attached PHY devices.
9–13	RA	Register address. Specifies one of up to 32 registers within the specified PHY device.
14–15	TA	Turnaround. Must be programmed to 10 to generate a valid MII management frame.
16–31	DATA	Management frame data. Field for data to be written to or read from PHY register.

To read or write on the MII management interface, MII_DATA is written by the user. To generate a valid read or write management frame, ST must be 01, OP must be 01 (management register write frame) or 10 (management register read frame), and TA must be 10.

To generate an 802.3-compliant MII management interface write frame (write to a PHY register) the user must write {01 01 PHYAD REGAD 10 DATA} to MII_DATA. Writing this pattern causes the control logic to shift data out of MII_DATA following a preamble generated by the control state machine. When the write management frame operation completes, the MII_DATAIO_COMPL interrupt is generated. At this time the contents of MII_DATA match the original value written.

To generate an MII management interface read frame (read a PHY register), the user must write {01 10 PHYAD REGAD 10 XXXX} to MII_DATA, (the content of the DATA field is a don’t care). Writing this pattern causes the control logic to shift data out of MII_DATA following a preamble generated by the control state machine. During this time, the contents of MII_DATA are serially shifted and are unpredictable if read by the user. An MII_DATAIO_COMPL interrupt is generated when the read management frame operation completes. At this time the contents of MII_DATA match the original value written except

for the DATA field, whose contents have been replaced by the value read from the PHY register.

Writing to MII_DATA during frame generation alters the frame contents. Software should use the MII_DATAIO_COMPL interrupt to avoid writing to the MII_DATA register during frame generation.

43.4.1.14 MII Speed Control Register (MII_SPEED)

The MII_SPEED register, shown in Figure 43-17, provides control of the MII clock (MDC pin) frequency and allows the MII management frame preamble to be dropped. MII_SPEED is written by the user.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xE84															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—								DIS_PREAMBLE	MII_SPEED						—
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xE86															

Figure 43-17. MII_SPEED Register

Table 43-20 describes MII_SPEED fields.

Table 43-20. MII_SPEED Field Descriptions

Bits	Name	Description
0–23	—	Reserved. Should be written to zero by the host processor.
24	DIS_PREAMBLE	Discard preamble. The MII standard allows the preamble to be dropped if the attached PHY devices does not require it. 0 Preamble is not discarded. 1 Preamble (32 1s) is not prepended to the MII management frame.
25–30	MII_SPEED	MII_SPEED controls the frequency of the MII management interface clock (MDC) relative to system clock. Clearing MII_SPEED, turns off the MDC and leaves it in low-voltage state. Any non-zero value generates an MDC frequency of 1/(MII_SPEED*2) of the system clock frequency.
31	—	Reserved. Should be written to zero by the host processor.

The MII_SPEED field must be programmed with a value to provide an MDC frequency of less than or equal to 2.5 MHz to comply with the IEEE MII specification. MII_SPEED must be non-zero to source a read or write management frame. After the management frame is

complete, MII_SPEED may optionally cleared to turn off the MDC. The MDC generated has a 50% duty cycle except when MII_SPEED is changed during operation (changes take effect following either a rising or falling edge of MDC).

If the system clock is 25 MHz, programming this register to 0x0000_000A generates an MDC frequency of 25 MHz * 1/10 = 2.5 MHz.

Table 43-21 shows optimum values for MII_SPEED as a function of system clock frequency.

Table 43-21. Programming Examples for MII_SPEED Register

System Clock Frequency	MII_SPEED[MII_SPEED]	MDC frequency
25 MHz	0x05	2.5 MHz
33 MHz	0x07	2.36 MHz
40 MHz	0x08	2.5 MHz
50 MHz	0x0A	2.5 MHz

43.4.1.15 FIFO Receive Bound Register (R_BOUND)

The R_BOUND register, Figure 43-18, is a read-only register the user can read to determine the upper address bound of the FIFO RAM. Drivers can use this value, along with the R_FSTART and X_FSTART to appropriately divide the available FIFO RAM between the transmit and receive data paths.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Reset	0000_0000_0000_0000															
R/W	Read only															
Addr	0xECC															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—					1	R_BOUND								—	
Reset	0000_0100_0000_0000															
R/W	Read only															
Addr	0xECE															

Figure 43-18. R_BOUND Register

Table describes R_BOUND fields.

Table 43-22. R_BOUND Field Descriptions

Bits	Name	Description
0–21	—	Reserved. Note all bits read back as 0 except for 21 which returns a 1.
22–29	R_BOUND	Read-only. Highest valid FIFO RAM address.
30–31	—	Reserved. Should be written to zero by the host processor.

43.4.1.16 FIFO Receive Start Register (R_FSTART)

The R_FSTART register, shown in Figure 43-19, is programmed by the user to indicate the starting address of the receive FIFO. R_FSTART marks the boundary between the transmit and receive FIFOs. The transmit FIFO uses addresses from X_FSTART to R_FSTART - 4. The receive FIFO uses addresses from R_FSTART to R_BOUND, inclusive.

Hardware initializes R_FSTART with a value that is microcode-dependent after ECNTRL[ETHER_EN] is set. R_FSTART only needs to be written to change the default value.

Figure 43-19. R_FSTART Register

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xED0															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—					1	R_FSTART								—	
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xED2															

Table 43-23 describes R_FSTART fields.

Table 43-23. R_FSTART Field Descriptions

Bits	Name	Description
0–21	—	Reserved. Note all bits read back as 0 except for 21 which returns a 1.
22–29	R_FSTART	Address of first receive FIFO location. Acts as a delimiter between receive and transmit FIFOs.
30–31	—	Reserved. Should be written to zero by the host processor.

43.4.1.17 Transmit Watermark Register (X_WMRK)

The X_WMRK register, shown in Figure 43-20, is used to control the amount of data required in the transmit FIFO before transmission of a frame can begin. This allows the user to minimize transmit latency (X_WMRK = 0x) or allow larger bus access latency (X_WMRK = 11) due to contention for the system bus. Setting the watermark to a high value lowers the risk of a transmit FIFO underrun due to system bus contention.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xEE4															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—														X_WMRK	
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xEE6															

Figure 43-20. X_WMRK Register

Table 43-24 bit field descriptions for X_WMRK.

Table 43-24. X_WMRK Field Descriptions

Bits	Name	Description
0–29	—	Reserved. Should be written to zero by the host processor.
30–31	X_WMRK	Transmit FIFO watermark. Frame transmission begins when the number of bytes selected by this field have been written into the transmit FIFO or if an end of frame has been written to the FIFO or if the FIFO is full before the selected number of bytes have been written. 0x 64 bytes written to the transmit FIFO 10 128 bytes written to the transmit FIFO 11 192 bytes written to the transmit FIFO

43.4.1.18 FIFO Transmit Start Register (X_FSTART)

The X_FSTART register, shown in Figure 43-21, can be programmed by the user to indicate the starting address of the transmit FIFO. X_FSTART is reset to the first available RAM address. The specific reset value is microcode-dependent. Users do not normally need to program X_FSTART. If users want to reserve RAM locations for other purposes, X_FSTART should never be set to value less than reset value.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xEEC															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—					1	X_FSTART								—	
Reset	0000_0					1	Microcode dependent								00	
R/W	Read/write															
Addr	0xEEE															

Figure 43-21. X_FSTART Register

Table 43-25 describes X_FSTART fields.

Table 43-25. X_FSTART Field Descriptions

Bits	Name	Description
0–21	—	Reserved. Note that all bits read back as 0 except for 21 which returns a 1.
22–29	X_FSTART	Address of first transmit FIFO location.
30–31	—	Reserved. Should be written to zero by the host processor.

43.4.1.19 DMA Function Code Register (FUN_CODE)

The FUN_CODE register, shown in Figure 43-22, contains the function code and byte order fields to be used during each transfer between the DMA and the SDMA interface. These bits can be written/read by the user.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—	DATA_BO0	DATA_BO1	DESC_BO0	DESC_BO1	FC1	FC2	FC3	—							
Reset	Undefined															
R/W	Read/write															
Addr	0xF34															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—															
Reset	Undefined															
R/W	Read/write															
Addr	0xF36															

Figure 43-22. FUN_CODE Register

Table 43-26 describes FUN_CODE fields.

Table 43-26. FUN_CODE Field Descriptions

Bits	Name	Description
0	—	Reserved. This bit reads as zero.
1–2	DATA_BO	Byte order. Supplied to the SDMA interface during receive and transmit data DMA transfers. 0x Reserved 1x Big-endian (Motorola) or true little-endian (DEC or Intel) byte ordering. Considering each word in the buffer, data bytes are received or transmitted from address 0b00 to 0b11. This is because communication is byte-oriented, and byte reads and writes are identical in big- and little-endian modes
3–4	DESC_BO	The byte order field supplied to the SDMA interface during receive and transmit open descriptor DMA transfers, and during close descriptor DMA transfers. 0x Reserved 1x Big-endian (Motorola) or true little-endian (DEC or Intel) byte ordering. Considering each word in the buffer, data bytes are received or transmitted from address 0b00 to 0b11. [This is because reception or transmission in communications is byte-oriented and byte reads and writes are identical in big-endian and little-endian modes].
5–7	FC	The function code field supplied to the SDMA interface during all DMA transfers.
8–31	—	Reserved. These bits read as zero.

43.4.1.20 Receive Control Register (R_CNTRL)

The R_CNTRL register, shown in Figure 43-23, is programmed by the user to control the operational mode of the receive block.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xF44															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—											BC_REJ	PROM	MII_MODE	DRT	LOOP
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xF46															

Figure 43-23. R_CNTRL Register

Table 43-27 describes R_CNTRL fields.

Table 43-27. R_CNTRL Field Descriptions

Bits	Name	Description
0–26	—	Reserved. This bit reads as zero.
27	BC_REJ	Broadcast frame reject. If set, frames with DA + 0xFFFF_FFFF_FFFF are rejected unless the PROM bit set. If both BC_REJ and PROM = 1, frames with broadcast DA are accepted and RxBD[M] is set.
28	PROM	Promiscuous mode. 0 Promiscuous mode disabled 1 Promiscuous mode enabled. All frames are accepted regardless of address matching.
29	MII_MODE	Selects external interface mode for both transmit and receive blocks. 0 Selects seven-wire mode (used only for serial 10 Mbps) 1 Selects MII mode.
30	DRT	Disable receive on transmit. 0 Receive path operates independently of transmit (use for full duplex or to monitor transmit Selects seven-wire mode (used only for serial 10 Mbps) 1 Disable reception of frames while transmitting (normally used for half-duplex mode)
31	LOOP	Internal loopback. If set, transmitted frames are looped back internal to the device and the transmit output signals are not asserted. The system clock is substituted for the TX_CLK when LOOP is asserted. DRT must be 0 when asserting LOOP.

43.4.1.21 Receive Hash Register (R_HASH)

R_HASH[MAX_FRAME_LENGTH], shown in Figure 43-24, is programmable. This field lets the user set the frame length (in bytes) at which the BABR and BABT interrupts and RxBD[LG] should be set.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xF48															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—					MAX_FRAME LENGTH										
Reset	0000_0101_1110_1110															
R/W	Read/write															
Addr	0xF4A															

Figure 43-24. R_HASH Register

Table 43-27 describes R_HASH fields.

Table 43-28. R_HASH Field Descriptions

Bits	Name	Description
0–7	—	Reserved for internal use. When read, these bits are unpredictable.
8–20	—	Reserved. These bits are read as zeros.
21–31	MAX_FRAME_LENGTH	User read/write field. Resets to decimal 1518. Length is measured starting at DA and includes the CRC at the end of the frame. Transmit frames longer than MAX_FRAME_LENGTH cause an BABT interrupt. Receive frames longer than MAX_FRAME_LENGTH cause a BABR interrupt and set the LG bit in the end-of-frame BD. The recommended value to be programmed by the user is 1518 or 1522 (if VLAN tags are supported).

43.4.1.22 Transmit Control Register (X_CNTRL)

The transmit control register (X_CNTRL), shown in Figure 43-25, is written by the user to configure the transmit block.

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xF84															
Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—													FDEN	HBC	GTS
Reset	0000_0000_0000_0000															
R/W	Read/write															
Addr	0xF86															

Figure 43-25. X_CNTRL Register

Table 43-29 describes X_CNTRL fields.

Table 43-29. X_CNTRL Field Descriptions

Bits	Name	Description
0–28	—	Reserved. These bits read as zero.
29	FDEN	Full-duplex enable. If set, frames are transmitted independently of carrier sense and collision inputs. This bit should be modified only when ECNTRL[ETHER_EN] is cleared.
30	HBC	Heartbeat control. If HBC = 1 and FDEN = 0, the heartbeat check is performed after transmission and TxBD[HB] and IEVENT[HBERR] are set, if the collision input does not assert within the heartbeat window. HBC should be modified only when ECNTRL[ETHER_EN] is cleared.
31	GTS	Graceful transmit stop. When GTS is set, the MAC stops transmission after any frame being transmitted is complete and INTR_EVENT[GRA] is set. If frame transmission is not underway, the GRA interrupt is asserted immediately. When transmission completes, clearing GTS causes the next frame in the transmit FIFO to be sent. If an early collision occurs during transmission when GTS = 1, transmission stops after the collision. The frame is sent again once GTS is cleared. Note that there may be old frames in the transmit FIFO that are sent when GTS is reasserted. To avoid this, clear ECNTRL[ETHER_EN] after the GRA interrupt.

43.4.2 Initialization Sequence

This section describes which registers and RAM locations are reset due to hardware reset, which are reset due to the microcontroller, and which locations the user must initialize before enabling the FEC.

43.4.2.1 Hardware Initialization

In the FEC, only registers that generate interrupts to the MPC8xx processor or cause conflict on bidirectional buses are reset by hardware. The registers shown in Table 43-30 are reset due to a hardware reset.

Table 43-30. Hardware Initialization

User/System	Register/Machine	Reset Value
User	ECNTRL	Cleared
User	IEVENT	Cleared
User	IMASK	Cleared
User	MII.SPEED	Cleared
User	PORT DPAR	Cleared
User	PORT DIR	Cleared

Other registers are reset whenever ECNTRL[ETHER_EN] is cleared. Clearing ETHER_EN immediately stops all DMA accesses and stops transmit activity after a bad CRC is sent; refer to Table 43-31.

Table 43-31. ECNTRL[ETHER_EN] Deassertion Initialization

User/System	Register/Machine	Reset Value
User	R_DES_ACTIVE	Cleared
User	X_DES_ACTIVE	Cleared

43.4.2.2 User Initialization (before Setting ECNTRL[ETHER_EN])

The user must initialize portions of the FEC before setting ECNTRL[ETHER_EN]. The exact values depend on the application. The sequence resembles that shown in Table 43-32.

Table 43-32. User Initialization (before Setting ECNTRL[ETHER_EN])

Step	Description
1	Set IMASK
2	Clear IEVENT
3	Set IVEC (define ILEVEL)
4	Set R_FSTART (optional)
5	Set X_FSTART (optional)
6	Set ADDR_HIGH and ADDR_LOW
7	Set HASH_TABLE_HIGH and HASH_TABLE_LOW
8	Set R_BUFF_SIZE
9	Set R_DES_START
10	Set X_DES_START
11	Set R_CNTRL
12	Set X_CNTRL
13	Set FUN_CODE

Table 43-32. User Initialization (before Setting ECNTRL[ETHER_EN]) (continued)

Step	Description
14	Set MII_SPEED (optional)
15	Initialize (empty) TxBD ring
16	Initialize (empty) RxBD ring
17	Set Port D PDPAR register
18	Set Port D PDDIR register

43.4.2.2.1 Descriptor Controller Initialization

In the FEC, the descriptor control machine initializes a few registers whenever ECNTRL[ETHER_EN] is set. The transmit and receive FIFO pointers are reset, the transmit backoff random number is initialized and the transmit and receive blocks are activated. After the descriptor controller initialization sequence, hardware is ready for operation, waiting for R_DES_ACTIVE and X_DES_ACTIVE to be asserted by the user.

43.4.2.2.2 User Initialization (after Setting ECNTRL[ETHER_EN])

The user must initialize portions of the FEC after setting ECNTRL[ETHER_EN]. The exact values depend on the application. The sequence resembles that shown in Table 43-33. (though these steps could also be done before setting ETHER_EN).

Table 43-33. User Initialization (after Setting ECNTRL[ETHER_EN])

Step	Description
1	Fill RxBD ring with empty buffers
2	Set R_DES_ACTIVE

43.4.3 Buffer Descriptors (BDs)

Data for Fast Ethernet frames must reside in external memory. Frame data is placed in one or more buffers, each of which is pointed to by a BD, which also contains the current state of the buffer. For maximum user flexibility, BDs are also located in external memory.

A buffer is produced by setting TxBD[R] or RxBD[E]. Writing to either X_DES_ACTIVE or R_DES_ACTIVE indicates that a buffer is in external memory for the transmit or receive data traffic, respectively. The hardware reads the BDs and processes the buffers. After the DMA transfer of the data and the updating of the BD status bits, hardware clears TxBD[R] or RxBD[E] to signal that the buffer was processed. Software can poll the BDs or may rely on the buffer/frame interrupts to detect when buffers have been processed.

ECNTRL[ETHER_EN] operates as a reset to the BD/DMA logic. When ETHER_EN is cleared, the DMA engine BD pointers are reset to point to the starting TxBDs and RxBDs.

The BDs are not initialized by hardware during reset. At least one TxBD and one RxBD must be initialized by software (write 0x0000_0000 to the most significant word of the BD) before ETHER_EN is set.

The BDs operate as a ring. R_DES_START defines the starting address for the RxBD ring and X_DES_START defines the starting address for TxBD ring. The last BD in each ring is indicated by the wrap (W) bit. When set, W indicates that the next BD in the ring is at the location pointed to by R_DES_START and X_DES_START for the receive and transmit rings, respectively. BD rings must start on a double-word boundary.

43.4.3.1 Ethernet Receive Buffer Descriptor (RxBD)

Figure 43-26 shows the RxBD. The first word of the RxBD contains control and status bits. The user initializes RxBD[E,W] and the Rx buffer pointer. When the buffer has been accessed by a DMA, the FEC modifies RxBD[E,L,M,BC,MC,LG,NO,SH,CR,OV,TR] and writes the length of the used portion of the buffer in the first word. The FEC modifies RxBD[M,BC,MC,LG,NO,SH,CR,TR,OV] only if L = 1.

Figure 43-26. Receive Buffer Descriptor (RxBD)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
+0	**E**	RO1	**W**	RO2	L	0	0	M	BC	MC	LG	NO	SH	CR	OV	TR
+2	DATA LENGTH															
+4	**RX BUFFER POINTER A[0–15]**															
+6	**RX BUFFER POINTER A[16–31]**															

The RxBD format is shown in Table 43-34.

Table 43-34. Receive Buffer Descriptor (RxBD) Field Description

Bits	Name	Description
0	**E**	Empty. Written by the FEC and user. Note that if the software driver sets RxBD[E], it should then write to R_DES_ACTIVE. 0 The buffer associated with this BD is filled with received data, or reception was aborted due to an error. The status and length fields have been updated as required. 1 The buffer associated with this BD is empty, or reception is in progress.
1	RO1	Receive software ownership bit. Software use. This read/write bit is modified by hardware and does not affect hardware.
2	**W**	Wrap, written by user. 0 The next BD is found in the consecutive location 1 The next BD is found at the location defined in RAM.R_DES_START.
3	RO2	Receive software ownership bit. Software use. This read/write bit is not modified by hardware and does not affect hardware.
4	L	Last in frame, written by FEC. 0 The buffer is not the last in a frame. 1 The buffer is the last in a frame.
5–6	—	Reserved.

Table 43-34. Receive Buffer Descriptor (RxBD) Field Description (continued)

Bits	Name	Description
7	M	Miss, written by FEC.Set by the FEC for frames that were accepted in promiscuous mode but were flagged as a miss by the internal address recognition. Thus, while promiscuous mode is being used, the user can use the M bit to quickly determine whether the frame was destined to this station. This bit is valid only if both the L bit and PROM bit are set. 0 The frame was received because of an address recognition hit. 1 The frame was received because of promiscuous mode.
8	BC	Set if the DA is broadcast.
9	MC	Set if the DA is multicast and not broadcast.
10	LG	Rx frame length violation, written by FEC. The frame length exceeds the value of MAX_FRAME_LENGTH in the bytes. The hardware truncates frames exceeding 2047 bytes so as not to overflow receive buffers This bit is valid only if the L bit is set.
11	NO	Rx nonoctet-aligned frame, written by FEC. A frame that contained a number of bits not divisible by 8 was received and the CRC check that occurred at the preceding byte boundary generated an error. NO is valid only if the L bit is set. If this bit is set the CR bit is not set.
12	SH	Short frame, written by FEC. A frame length that was less than the minimum defined for this channel was recognized.Note that the MPC860T does not support SH, which is always zero.
13	CR	Rx CRC error, written by FEC. This frame contains a CRC error and is an integral number of octets in length. This bit is valid only if the L bit is set.
14	OV	Overrun, written by FEC. A receive FIFO overrun occurred during frame reception. If OV = 1, the other status bits, M, LG, NO, SH, CR, and CL lose their normal meaning and are cleared. This bit is valid only if the L bit is set.
15	TR	Truncate. Set if the receive frame is truncated (≥ 2 Kbytes).
Offset+2	Data length	Data length, written by FEC. Data length is the number of octets written by the FEC into this BD's buffer if L = 0 (the value = R_BUFF_SIZE), or the length of the frame including CRC if L = 1. It is written by the FEC once as the BD is closed.
Offset+4	**Rx buffer pointer**	Rx buffer pointer A[0–31], written by user. The receive buffer pointer, which always points to the first location of the associated buffer, must always be a multiple of 16. The buffer must reside in memory external to the FEC.

43.4.3.2 Ethernet Transmit Buffer Descriptor (TxBD)

Data is presented to the FEC for transmission by arranging it in buffers referenced by the channel's TxBDs. The FEC confirms transmission or indicates error conditions using BDs to inform the host that the buffers have been serviced. The user initializes TxBD[R,W,L,TC], the length (in bytes), and the buffer pointer.

- If L = 0, the FEC clears the R bit when the buffer is accessed. Status bits are not modified.
- If L = 1, the FEC clears the R bit and modifies the DEF, HB, LC, RL, RC, UN, and CSL status bits after the buffer is accessed and frame transmission completes.

The TxBD is shown in Figure 43-27.

Table 43-35 describes TxBD fields.

Figure 43-27. Transmit Buffer Descriptor (TxBD)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
+0	**R**	TO1	**W**	TO2	**L**	**TC**	DEF	HB	LC	RL	RC				UN	CSL
+2	**DATA LENGTH**															
+4	**Tx Data Buffer Pointer A[0–15]**															
+6	**Tx Data Buffer Pointer A[16–31]**															

Table 43-35. Transmit Buffer Descriptor (TxBD) Field Descriptions

Bits	Name	Description
0	**R**	Ready, written by FEC and user. 0 The buffer associated with this BD is not ready for transmission. The user can manipulate this BD or its associated buffer. The FEC clears R after the buffer is sent or an error occurs. 1 The user-prepared buffer has not been sent or is being sent. The user cannot update the BD while R = 1.
1	TO1	Transmit software ownership bit. This field is available for use by software. This read/write bit is not modified by hardware and its value does not affect hardware.
2	**W**	Wrap, written by user. 0 The next BD is found in the consecutive location 1 The next BD is found at the location defined in X_DES_START.
3	TO2	Transmit software ownership bit This field is available for use by software. This read/write bit is not modified by hardware and its value does not affect hardware.
4	**L**	Last in frame, written by user. 0 The buffer is not the last in the transmit frame. 1 The buffer is the last in the transmit frame.
5	**TC**	Tx CRC, written by user (valid if L = 1). 0 End transmission immediately after the last data byte. 1 Transmit the CRC sequence after the last data byte.
6	DEF	Defer indication, written by FEC (valid if L = 1). Set when the FEC had to defer while trying to transmit a frame. This bit is not set if a collision occurs during transmission.
7	HB	Heartbeat error, written by FEC (valid if L = 1). Set to indicate that the collision input was not asserted within the heartbeat window after transmission completed. HB can be set only if X_CNTRL[HBC] = 1.
8	LC	Late collision, written by FEC (valid if L = 1). Set to indicate that a collision occurred after 56 data bytes were transmitted. The FEC terminates the transmission.
9	RL	Retransmission limit, written by FEC (valid if L = 1). Set to indicate that the transmitter failed retry limit + 1 attempts to send a message due to repeated collisions.
10–13	RC	Retry count, written by FEC (valid if L = 1). Counts retries needed to successfully send this frame. If RC = 0, the frame was sent correctly the first time. If RC = 15, the frame was sent successfully while the retry count was at its maximum value. If RL = 1, RC has no meaning.
14	UN	Underrun, written by FEC (valid if L = 1). If set, the FEC encountered a transmit FIFO underrun while sending one or more buffers associated with this frame. When a Tx FIFO underrun occurs, transmission of the frame stops and an incorrect CRC is appended. Any remaining buffers associated with this frame are accessed and dumped by the transmit logic.
15	CSL	Carrier sense lost, written by FEC (valid if L = 1). Carrier sense dropped out or never asserted during transmission of a frame without collision.

Table 43-35. Transmit Buffer Descriptor (TxBD) Field Descriptions (continued)

Bits	Name	Description
Offset+2	**Data length**	Data length, written by user and never by the FEC. Indicates the number of octets the FEC should send from this BD's buffer. The DMA engine uses bits 21–31. Bits 16–20 are ignored.
Offset+4	**Tx buffer pointer**	Tx buffer pointer A[0–31], written by user and never by the FEC. The transmit buffer pointer, which contains the address of the associated buffer, may be even or odd. The buffer must reside in external memory.

On transmit, an underrun occurs if the transmit FIFO empties of data before the end of the frame. In this case, a bad CRC is appended to the partially transmitted data. In addition, the UN bit is set in the last BD in the current frame. This situation can occur if the FEC cannot access the 60x bus or if the next BD in the frame is unavailable.

NOTE

A software driver that sets TxBD[R] should then write to X_DES_ACTIVE.

Part VIII
System Debugging and Testing Support

Intended Audience

Part VIII is intended for system designers who need to test and debug their MPC855T design.

Contents

Part VIII describes how to use the MPC855T facilities for debugging and system testing. It contains the following chapters:

- Chapter 44, “System Development and Debugging,” describes support provided for program flow tracking, internal watchpoint and breakpoint generation, and emulation systems control.
- Chapter 45, “IEEE 1149.1 Test Access Port,” describes the dedicated user-accessible test access port (TAP), which is fully compatible with the *IEEE 1149.1 Standard Test Access Port and Boundary Scan Architecture*.

Suggested Reading

This section lists additional reading that provides background for the information in this manual.

MPC8xx Documentation

Supporting documentation for the MPC855T can be accessed through the world-wide web at http://www.motorola.com. This documentation includes technical specifications, reference materials, and detailed application notes.

Conventions

This document uses the following notational conventions:

Bold	Bold entries in figures and tables showing registers and parameter RAM should be initialized by the user.

mnemonics	Instruction mnemonics are shown in lowercase bold.
italics	Italics indicate variable command parameters, for example, **bcctr***x*. Book titles in text are set in italics.
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote binary number
REG[FIELD]	Abbreviations or acronyms for registers or buffer descriptors are shown in uppercase text. Specific bits, fields, or numerical ranges appear in brackets. For example, MSR[LE] refers to the little-endian mode enable bit in the machine state register.
x	In certain contexts, such as in a signal encoding or a bit field, indicates a don't care.
n	Indicates an undefined numerical value

Acronyms and Abbreviations

Table i contains acronyms and abbreviations used in this document. Note that the meanings for some acronyms (such as SDR1 and DSISR) are historical, and the words for which an acronym stands may not be intuitively obvious.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
BIST	Built-in self test
CPM	Communication processor module
IEEE	Institute of Electrical and Electronics Engineers
JTAG	Joint Test Action Group
LSB	Least-significant byte
lsb	Least-significant bit
LSU	Load/store unit
MSB	Most-significant byte
msb	Most-significant bit
Rx	Receive
SPR	Special-purpose register
TAP	Test access port
Tx	Transmit

Chapter 44
System Development and Debugging

Emulators require a level of control and observation that is 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 help development tools support, some development support functions are implemented in the silicon. Program flow tracking, internal watchpoint and breakpoint generation, and emulation systems control over the activity of the core (debug mode) are some of the features that allow the user to efficiently debug MPC855T-based systems.

44.1 Tracking Program Flow

The MPC855T provides many options for tracking program flows that impact performance in varying degrees.

- In one mode, signals provided for tracking code flow can be captured externally and then parsed by a post-processing program. This mode is described more fully in subsequent sections.
- In another, slower mode, instruction flow is visible on the external bus when the MPC855T is programmed to operate in serialized mode with all fetch cycles shown on the external bus. Although instruction flow tracking is simpler, performance is much lower than in regular mode. Section 44.5.1.3, “Instruction Support Control Register (ICTRL),” describes programming of the core to operate in this mode.

The MPC855T 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 I-cache), increase performance but make it very difficult to provide the user 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 architecturally executed.

To reconstruct program trace, the program code, combined with additional MPC855T information, is required. Reporting program trace during retirement significantly complicates the implementation in two ways: 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 deciphered by monitoring fetched code and instruction queue flushes, and using this information to reconstruct which instructions actually reach retirement. 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 may be architecturally executed or they may be canceled in some stage of the machine pipeline.

The information required to enable reconstruction of program trace includes:

- A description of the last fetched instruction (stall, sequential, branch not taken, branch direct taken, branch indirect taken, interrupt/exception taken).
- The addresses of the targets of all indirect flow changes. Indirect flow changes include all branches using the link and count registers as the target address, all interrupts/exceptions, and **rfi** and **mtmsr** (because they may cause context switches).
- The number of instructions canceled on each clock.

The following sections define how this information is generated and how it should be used to reconstruct the program trace.

44.1.1 Program Trace Functional Description

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 resulting from indirect flow changes. When program trace recording is required, the user can ensure these cycles are visible on the external bus.

The core can be forced to show all fetch cycles marked with the program trace cycle attribute either by setting TECR[VSYNC] of the development port or by programming ISCT_SER in the instruction support control register (ICTRL). For more information on VSYNC see **Section 44.3.2, “Development Port Communication**.” Both states described here are subsequently referred to as VSYNC state.

The VSYNC state 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, entering VSYNC state forces the machine to synchronize and the first fetch after this synchronization to be marked as a program trace cycle and be seen on the external bus.

In VSYNC state, 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 an external device or generate address-only cycles when instructions are in internal devices (I-cache and internal memory). In VSYNC state, performance degrades because of the additional external bus cycles. However, this degradation is very small.

Note that program trace functions are not available when operating the MPC855T in half-speed bus mode (when SCCR[EBDF] = 0b01). The VFLS[0–1] signals are not valid in half-speed bus mode.

44.1.2 Instruction Fetch Show Cycle Control

Instruction fetch show cycles are controlled by ICTRL[ISCT_SER] and the state of VSYNC. Table 44-1 defines the level of fetch show cycles generated by the core.

Table 44-1. Fetch Show Cycles Control

VSYNC	ICTRL[ISCT_SER] Instruction Fetch Show Cycle Control Bits	Show Cycles Generated
X	000	All fetch cycles
X	X01	All change of flow (direct and indirect)
X	X10—Enable $\overline{\text{STS}}$ functionality of OP2/MODCK1/$\overline{\text{STS}}$ by writing 10 or 11 to SIUMCR[DBGC]. The external bus address should be sampled only when $\overline{\text{STS}}$ is asserted.	All indirect change of flow
0	X11	No show cycles are performed
1	X11	All indirect change of flow

A cycle marked with the program trace cycle attribute is generated when entering and exiting VSYNC state by setting TECR[VSYNC].

44.1.3 Program Trace Signals

Note that if the MPC855T is in half-speed bus mode (SCCR[EBDF] = 0b01), the VF and VFLS pins do not report fetch and flush information for the program trace capability. However, the internal freeze state of the processor is reported in the VFLS pins as it does in full-speed bus mode. The status pins are divided into two groups, shown in Table 44-2.

Table 44-2. Status Pin Groupings

Pins	Description
VF [0–2]	Instruction queue status. Denotes the type of the last fetched instruction or how many instructions were flushed from the instruction queue. VF [0–2] are used for both functions because queue flushes occur only in clocks in which no fetch type information is reported. Table 44-3 defines instruction queue flushes and Table 44-4 defines instruction fetch types.
VFLS [0–1]	History buffer flushes status: indicates the number of instructions that are flushed from the history buffer on this clock. Possible values are as follows: 00 None 01 1 instruction was flushed from the history buffer 10 2 instructions were flushed from the history buffer 11 Used for debug mode indication. Should be ignored by the program trace external hardware. See Section 44.3.1, “Debug Mode Operation.”

Table 44-3 describes possible instruction queue flushes as they are identified by VF encodings.

Table 44-3. VF Pins Encoding: Instruction Queue Flushes

VF	Instructions Flushed	VF Next Cycle Holds
000	None	Instruction type information
001	One instruction was flushed from the instruction queue	Instruction type information
010	Two instructions were flushed from the instruction queue	Instruction type information
011	Three instructions were flushed from the instruction queue	Instruction type information
100	Four instructions were flushed from the instruction queue	Instruction type information
101	Five instructions were flushed from the instruction queue	Instruction type information
110	Reserved	Instruction type information
111	See VF = 0b111 entry in Table 44-4	—

Table 44-4 describes instruction types as they are identified by VF encodings.

Table 44-4. VF Pins Encoding: Instruction Fetch Types

VF	Instruction Type	VF Next Clock Holds
000	None	More instruction type information
001	Sequential [1]	
010	Branch (direct or indirect) not taken	
011	This instruction is marked with the program trace cycle attribute in response to changing the state of TECR[VSYNC] in the development port.	
100	Interrupt/exception taken, the target is 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 is marked with the program trace cycle attribute[1]	
110	Branch direct taken	
111	Branch (direct or indirect) not taken	

1 See **Section 44.1.4.3, "Sequential Instructions Marked as Indirect Branch**."

2 Unless the next clock VF = 111, see **Section 44.1.4.1, "Queue Flush Information Special Case**."

44.1.4 Program Trace Special Cases

The following sections describe special cases of program trace implemented on the MPC855T.

44.1.4.1 Queue Flush Information Special Case

There is one special case where the queue flush information is expected on the VF pins after an instruction fetch encoding of VF = 0b1xx. This case is where an instruction-type VF indications of b1xx is followed by an indication of VF = 0b111. THis indication of

VF = 0b111 should be interpreted as an instruction fetch type encoding, as described by Table 44-4. 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.

44.1.4.2 Program Trace When In Debug Mode

When entering debug mode an interrupt/exception taken is reported on the VF pins (VF = 0b100) and a cycle marked with the program trace cycle is made externally visible. When the core is in debug mode, VF = 0b000 and VFLS = 0b11. For more information on debug mode, see Section 44.3, “Development System Interface.”

If TECR[VSYNC] is set or cleared while the core is in debug mode, this information is reported 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 = 0b101 (indirect branch) due to the **rfi** instruction that is being issued. In both cases, the first instruction fetch after debug mode is marked with the program trace cycle attribute and is externally visible.

44.1.4.3 Sequential Instructions Marked as Indirect Branch

There are instances where non-branch (sequential) instructions can affect the machine in a manner similar to indirect branch instructions. These instructions include **rfi**, **mtmsr**, **isync**, and **mtspr** to registers CMPA–CMPF, ICTRL, ICR, and DER.

The core marks these instructions are marked as indirect branch instructions (VF = 0b101). The next instruction address is marked with the program trace cycle attribute, as if it were an indirect branch target. Therefore, when one of these special instructions is detected in the core, the address of the next instruction is externally visible. The reconstructing software can now correctly evaluate the effect of these instructions.

44.1.5 Reconstructing Program Trace

When program trace is needed, 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. Although program trace can be used in various ways, the following describes only back trace and window trace.

44.1.5.1 Back Trace

Back trace is useful when a record of the program trace before an event occurred is needed. An example of such an event is a system failure. If back trace is needed, external hardware should start sampling VF and VFLS and the address of all cycles marked with the program trace cycle attribute immediately after reset is negated.

At reset, cycles marked with the program trace cycle attribute are visible on the external bus (that is, the instruction fetch show cycle/core serialize control field (ICTRL[ISCT_SER])

is cleared at reset). To avoid this slower default mode, it is recommended that the user enters VSYNC state as described in Section 44.1.1, “Program Trace Functional Description.” To exit VSYNC state after a particular event, either trap in debug mode and trigger the freeze indication or follow the method described in Section 44.1.1, “Program Trace Functional Description.” After exiting VSYNC state, the trace buffer holds the trace of the program executed before the pertinent event occurred.

44.1.5.2 Window Trace

Window trace is useful when a record of the program trace between two events is needed, in which case, VSYNC state should be entered between these two events. After exiting VSYNC state, the trace buffer holds trace information for the program executed between the two events.

44.1.5.2.1 Synchronizing the Trace Window to Internal Core Events

The assertion/ negation of VSYNC is accomplished using the serial interface implemented in the development port. To synchronize the assertion/negation of VSYNC to an internal event of the core, it is possible to use the internal breakpoints hardware with the debug mode. This method is available only when debug mode is enabled. For more information on debug mode, see Section 44.3, “Development System Interface.”

The following is a possible set of steps that enable the user to synchronize the trace window to the internal core events:

1. Enter debug mode, either immediately out of reset or using the debug mode request.
2. Program hardware to break on the event that marks the start of the trace window using the registers defined in Section 44.2, “Watchpoints and Breakpoints Support.”
3. Enable debug mode entry for the breakpoint programmed in the DER (see Table 44-25).
4. Return to the regular code run (refer to Section 44.3.1.7, “Exiting Debug Mode”).
5. The hardware generates a breakpoint when the event in question is detected and the machine enters debug mode (refer to Section 44.3.1.2, “Entering Debug Mode”).
6. Program the hardware to break on the event that marks the end of the trace window.
7. Assert VSYNC.
8. Return to the regular code run. The first report on the VF pins is VSYNC (VF = 0b011).
9. The external hardware starts sampling the program trace information after the VF pins indicate VSYNC.
10. The hardware generates a breakpoint when the event in question is detected and the machine enters debug mode.

11. Negate VSYNC.
12. Return to the regular code run (issue an **rfi**). The first encoding on the VF pins is VSYNC (VF = 0b011).
13. External hardware stops sampling the program trace information after recognizing VSYNC on the VF pins.

44.1.5.3 Detecting the Trace Window Start Address

When using back trace, latching of VF, VFLS, and the address of the cycles marked program trace cycle should all start immediately after the negation of reset. 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 Table 44-5 to calculate the trace window start address.

Table 44-5. Detecting the Trace Buffer Start Point

VF1	VF2	Starting Point	Description
011 VSYNC	001 Sequential	T1	VSYNC asserted. Followed by a sequential instruction. The start address is T1.
011 VSYNC	110 Branch direct taken	T1 - 4 + Offset(T1 - 4)	VSYNC asserted. Followed by a taken direct branch. The start address is the target of the direct branch.
011 VSYNC	101 Branch indirect taken	T2	VSYNC asserted. Followed by a taken indirect branch. The start address is the target of the indirect branch.

44.1.5.4 Detecting the Assertion/Negation of VSYNC

Since the VF pins are used for reporting both instruction type and queue flush information, the external hardware must take special care when trying to detect entry and exit of the VSYNC state. When VF = 0b011, it is a VSYNC entry or exit report only if the prior value of VF was 0b000, 0b001, or 0b010.

44.1.5.5 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 (SIU) arbitrates over the external bus. Therefore, there is a delay between when a program trace cycle is reported as performed and the time that this cycle can be detected on the external bus.

When the user exits VSYNC state (through the serial interface of the development port), the core delays the report of VSYNC occurring 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 and should be ignored.

44.1.5.6 Efficient Trace Information Capture

To store all information generated on the pins during program trace (5 bits per clock + 30 bits per show cycle) a large memory buffer is required. However, because this information includes events that were canceled, some of this information can be discarded. External hardware can be added to eliminate all canceled instructions and report only on taken/not taken branches, indirect flow change, and the number of sequential instructions after the last flow change.

44.2 Watchpoints and Breakpoints Support

Watchpoints, when detected, are reported to the external world (on dedicated pins), but do not change the timing and flow of the machine. Breakpoints, when detected, force the machine to branch to the appropriate exception handler. The core supports watchpoints generated inside the core and breakpoints generated inside and outside the core.

Internal watchpoints are generated when a user-programmable set of conditions are met. Internal breakpoints can be programmed to be generated either as an immediate result of the assertion of one of the internal watchpoints 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 software, by setting the corresponding software trap enable bit or on-the-fly using the serial interface implemented in the development port to set the corresponding trap enable bit. External breakpoints can be generated by peripherals of the system outside of the MPC855T such as an external development system. Peripherals on the external bus use the serial interface of the development port to assert the external breakpoint.

In the core, as in other RISC processors, software saves and restores machine state as part of exception handling. As software saves/restores the machine state, MSR[RI] is cleared. Exceptions that occur are handled by the core when MSR[RI] is clear and they result in a nonrestartable machine state. See Section 6.1.5, “Recoverability after an Exception.”

In general, the core recognizes breakpoints only if MSR[RI] = 1, which guarantees machine restartability after a breakpoint. In this working mode, breakpoints are said to be masked. Sometimes it is preferable to enable breakpoints when MSR[RI] is clear, despite the risk of a nonrestartable machine state. Internal breakpoints also have a programmable nonmasked mode, and an external development system can choose to assert a nonmaskable external breakpoint. Watchpoints are not masked and are always reported on external pins, regardless of the value of MSR[RI]. Although they count watchpoints, counters are part of

the internal breakpoint logic and are not decremented when the core operates in masked mode and MSR[RI] = 0. Figure 44-1 shows the core's watchpoint and breakpoint support.

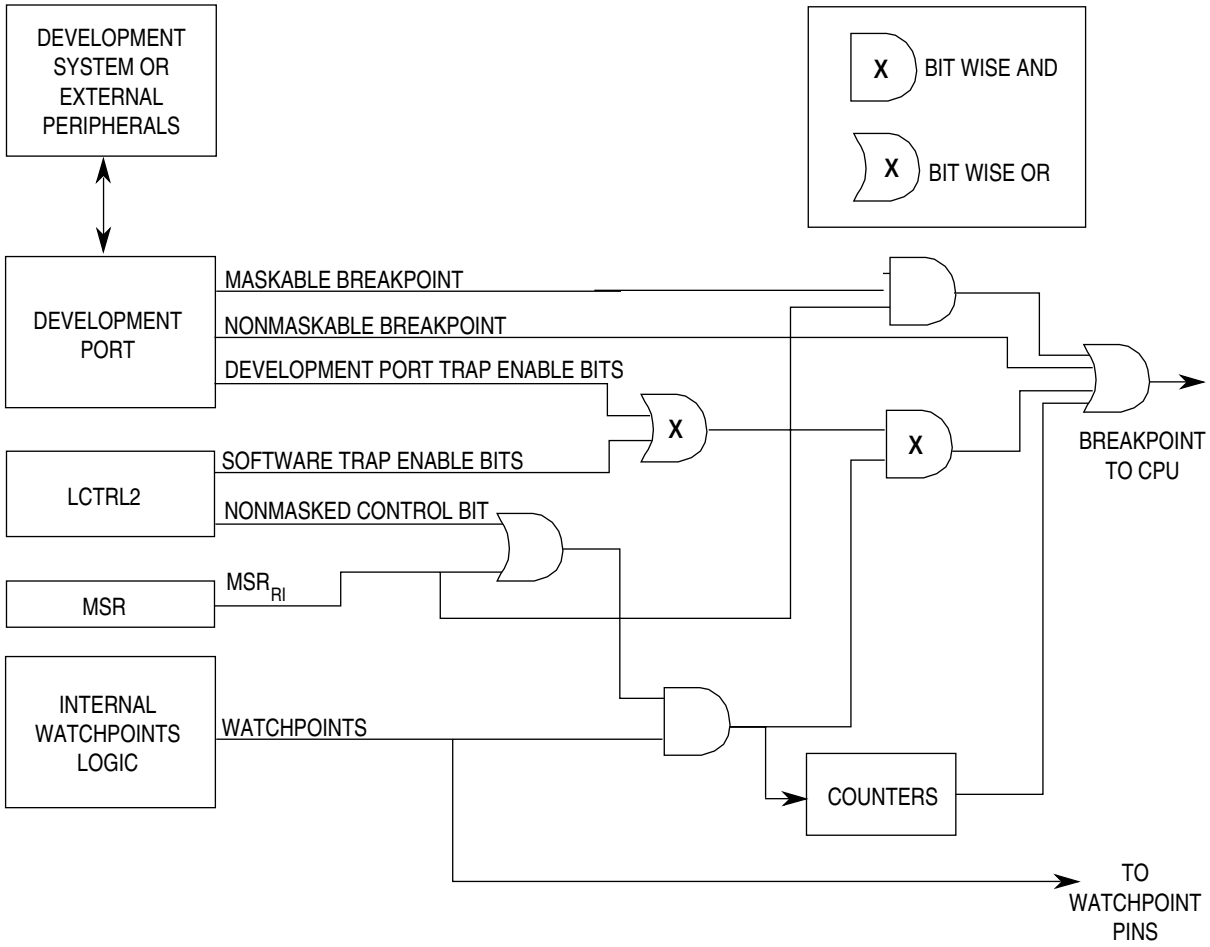


Figure 44-1. Watchpoints and Breakpoint Support in the Core

44.2.1 Key Features

The following list summarizes features of the internal watchpoints and breakpoints support.

- 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. Includes lsb masking, according to the size of the bus cycle for the byte and half-word working modes. See Section 44.2.4.2, "Byte and Half Word Working Modes."
- Two L-data comparators supporting equal, not equal, greater than, and less than. Includes byte, half-word, and word operating modes, and four byte mask bits for each comparator. It can be used for integer data. A match is detected only on the valid part of the data bus (according to the cycle's size and the two address lsbs).
- No internal breakpoint/watchpoint support for unaligned words and half words.

- L-data comparators can be programmed to treat integers as signed or unsigned.
- 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 instructions and two for loads/stores.
- Two dedicated 16-bit down counters. Each can count either an instruction watchpoint or load/store watchpoint. Only architecturally executed events are counted (count up is performed in case of recovery).
- On-the-fly trap enable programming of the different internal breakpoints using the development port serial interface (see **Section 44.3.2, “Development Port Communication**”). Software control is also available.
- Watchpoints do not change the timing of the machine.
- Internal breakpoints and watchpoints are detected on the instruction during 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 (due to exceptions or missed predictions) are not reported and do not change the machine’s timing.
- 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 BAR.
- Load/store multiple/string instructions with load/store breakpoints finish execution before the machine branches to the breakpoint exception routine.
- Load/store data compare is accomplished on the load/store, after swap in store accesses and before swap in load accesses (as the data appears on the bus).
- Internal breakpoints may operate either in masked mode or in nonmasked mode.
- “Go to x” and “continue” working modes are supported for instruction breakpoints.

44.2.2 Internal Watchpoints and Breakpoints Logic

Internal breakpoint and watchpoint support is based on the following:

- Eight comparators comparing information on instruction and load/store cycles
- Two counters
- Two AND-OR logic structures

Comparators perform compare on the instruction address (I-address), the load/store address (L-address), and the load/store data (L-data) and 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. See Section 44.2.4.5, “Generating Six Compare Types.” Using the AND-OR logic structures “in range” and “out of range” detections (on address and data) are supported. The counters can be used to program a breakpoint to be recognized after an event is detected a predefined number of times.

The L-data comparators operate on load or store integer data. When operating on integer data, L-data comparators perform comparisons on bytes, half-words, and words, treating numbers as signed or unsigned. Comparators generate match events, then instruction match events enter the instruction AND-OR logic where instruction watchpoints and breakpoints are generated. An asserted instruction watchpoint can generate an instruction breakpoint. Two different events can decrement one counter. When a counter on an instruction watchpoint expires, the instruction breakpoint is asserted.

Instruction watchpoints and load/store match events on address/data enter the load/store AND-OR logic where load/store watchpoints and breakpoints are generated. Load/store watchpoints (when asserted) can generate the load/store breakpoint or decrement a counter. When a counter on one load/store watchpoint expires, the load/store breakpoint is asserted.

Watchpoints progress in the machine and are reported on retirement. Internal breakpoints progress in the machine until they reach the top of the history buffer, at which point the machine branches to the breakpoint exception vector. To allow use of breakpoint features without restricting 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).

For more information, see Section 44.3, “Development System Interface.”

44.2.3 Functional Description

The following sections describe instruction and load/store watchpoint generation in detail.

44.2.3.1 Instruction Support Detailed Description

Each of the four instruction address comparators (A–D), shown in Figure 44-2, is 30 bits long and generates two output signals—equal and less than. These signals generate one of four events—equal, not equal, greater than, or less than. The instruction watchpoints and breakpoint are generated using these events according to the user programming. Using the OR option enables “out of range” detection.

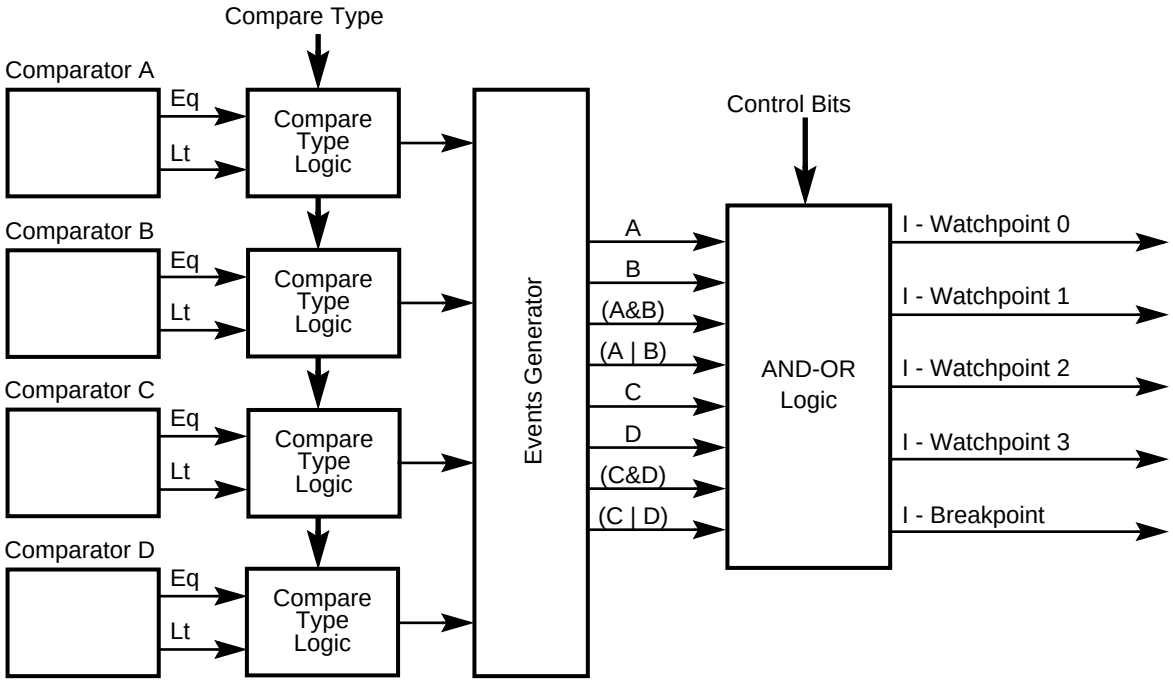


Figure 44-2. Instruction Support General Structure

Table 44-6 shows instruction watchpoint programming options.

Table 44-6. 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)

44.2.3.2 Load/Store Support Detailed Description

Each of the two load/store address comparators (E and F) compares the 32 address bits and the cycle’s attributes (read/write). The two lsbs are masked when a word is accessed; the

lsb is masked when a half word is accessed. As shown in Figure 44-3, each comparator generates two output signals—equal and less than. These signals generate one of four events from each comparator—equal, not equal, greater than, or less than. See Section 44.2.4.2, “Byte and Half Word Working Modes.”

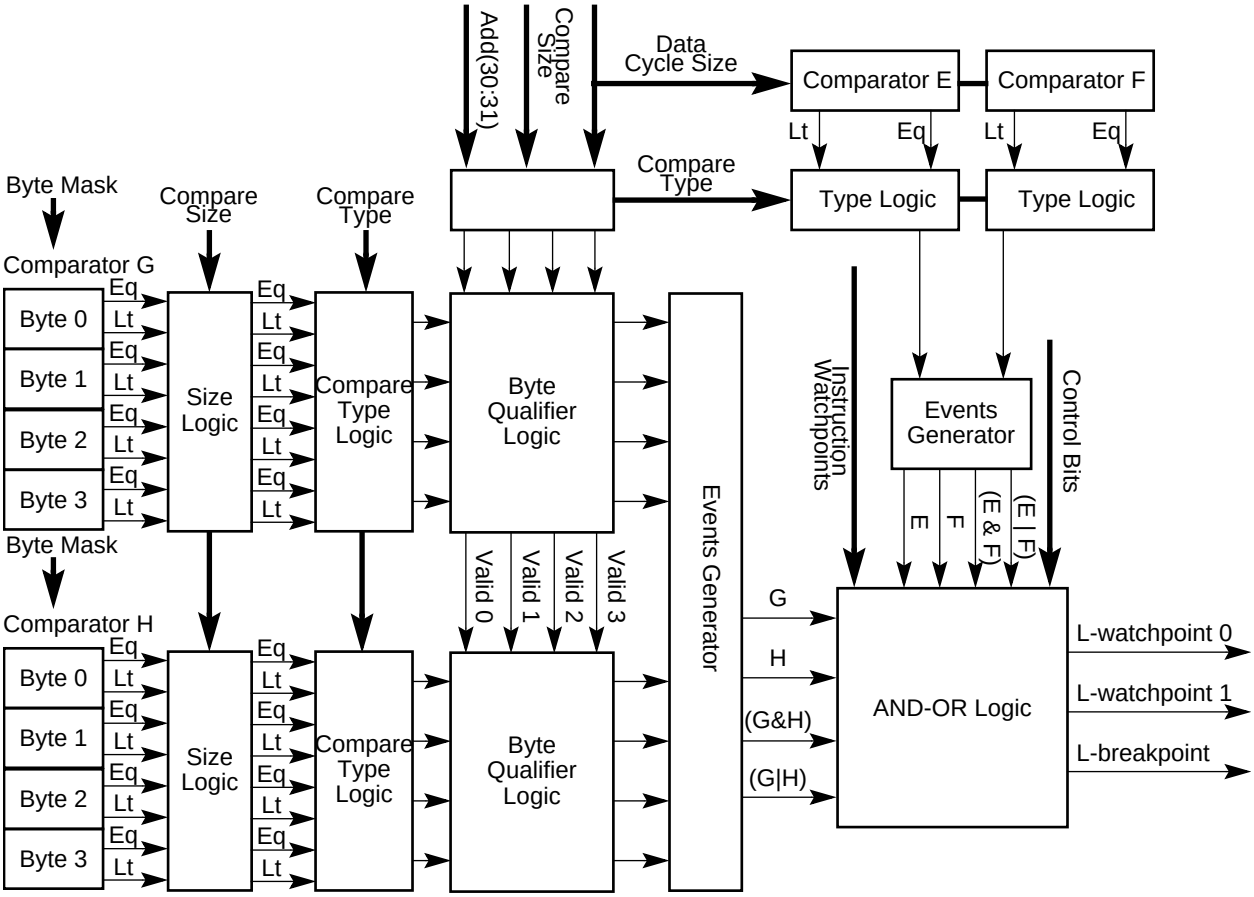


Figure 44-3. Load/Store Support General Structure

Each of the two load/store data comparators (G and H) is 32 bits wide and can be programmed to treat numbers as signed or unsigned. Each data comparator operates as four independent byte comparators. Each byte comparator has a mask bit and generates two output signals—equal and less than (if the mask bit is not set.) Therefore, each 32-bit comparator has eight output signals. These signals generate the equal and less-than signals according to the compare size programmed by the user (byte, half-word, word). All signals are significant in byte mode. In half-word mode, only four signals from each 32-bit comparator are significant; when operating in word mode, only two signals are significant.

One of the next four match events is generated by the equal and less-than signals—equal, not equal, greater than, or less than, depending on the compare type programmed. 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 is asserted. The match indication signals generate four load/store data events as shown in Table 44-7.

Table 44-7. Load/Store Data Events

Event Name	Event Function (see note)
G	(Gmatch0 | Gmatch1 | Gmatch2 | Gmatch3)
H	(Hmatch0 | Hmatch1 | Hmatch2 | Hmatch3)
(G & H) [1]	((Gmatch0 & Hmatch0) | (Gmatch1 & Hmatch1) | (Gmatch2 & Hmatch2) | (Gmatch3 & Hmatch3))
(G | H) [1]	((Gmatch0 | Hmatch0) | (Gmatch1 | Hmatch1) | (Gmatch2 | Hmatch2) | (Gmatch3 | Hmatch3))

[1] & denotes a logical AND. | 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 the user's programming.

Table 44-8. Load/Store Watchpoints Programming Options

Name	Description	Instruction Events Programming Options	L-Address Events Programming Options	L-Data Events Programming Options
LW0	First load/store watchpoint	IW0, IW1, IW2, IW3, ignore instruction events	Comparator E Comparator F Comparators (E & F) Comparators (E | F) Ignore L-address events	Comparator G Comparator H Comparators (G & H) Comparators (G | H) Ignore L-data events
LW1	Second load/store watchpoint	IW0, IW1, IW2, IW3, ignore instruction events	Comparator E Comparator F Comparators (E & F) Comparators (E | F) Ignore I-address events	Comparator G Comparator H Comparators (G & H) Comparators (G | H) Ignore L-data events

When programming load/store watchpoints to ignore L-addr events and L-data events, the instruction must be a load/store instruction for the load/store watchpoint event to trigger.

44.2.3.3 The Counters

Each of the two 16-bit down counters can count an instruction watchpoint or a load/store watchpoint. Both generate the corresponding breakpoint when they reach zero. In masked mode, counters do not count detected watchpoints when MSR[RI] = 0. See Section 44.2.4.3, "Context Dependent Filter." Counter values are not predictable if they count watchpoints programmed on instructions that alter counters directly. Readings from the counters when active must be synchronized by inserting a **sync** instruction before the read.

Note that when programmed to count instruction watchpoints, the last instruction that decrements the counter to zero is treated like any other instruction breakpoint in 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 one and not zero, as one might expect. 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 that it executes before the machine branches to the breakpoint exception routine. Therefore, the value of the counter inside the breakpoint exception routine equals zero.

44.2.3.4 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. See Section 44.3.2.4, “Development Port Serial Communications–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 ICTRL and the LCTRL2 using the **mtspr** instruction. Table 44-20 and Table 44-22 show the exact bit placement.

44.2.4 Operation Details

The following sections describe various operating details of watchpoint and breakpoint.

44.2.4.1 Restrictions

The same watchpoint can be detected more than once during execution of an instruction. For example, a load/store watchpoint can be detected on more than one transfer when executing load/store multiple/string instructions or a load/store watchpoint can be detected on more than one byte in byte mode. In such cases only one watchpoint of a given type is reported for the instruction. Similarly, only one watchpoint of the same type can be counted for a single instruction. Watchpoint events are reported when the instruction that caused the event retires; because more than one instruction can retire in a single clock, ensuing events may be reported in the same clock. Moreover, an event detected on more than one instruction (tight loops or range detection) can only be reported once. Internal counters count correctly in these cases.

44.2.4.2 Byte and Half Word Working Modes

The user 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, write the required match value to the correct half word of the data comparator and the mask in the L-data comparator. To break on bytes, the byte mask for each L-comparator and the bytes to be matched must be written in the data comparator.

Because bytes and half words can be accessed using a larger data width instruction, the user cannot 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 accessed using a load word instruction, the L-address value will be of the word (byte 0). Therefore, the core masks the two lsbs of the L-address comparators for word accesses and the lsb for half-word accesses. Address range is supported only when aligned according to access size.

44.2.4.2.1 Examples

The following examples show programming options for several search criteria:

- Example 1

  Looking for:
  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.
  Both byte masks = 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.

- Example 2

  Looking for:
  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.
  Both byte masks = 0x0.
  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.

- Example 3

  Looking for:
  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.
Both byte masks = 0x0.
Both L-data comparators program to half-word or word mode.
Result: An event is 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), false detections may occur.

- These can only be ignored by the software that handles the breakpoints. Figure 44-4 shows this partially supported scenario:

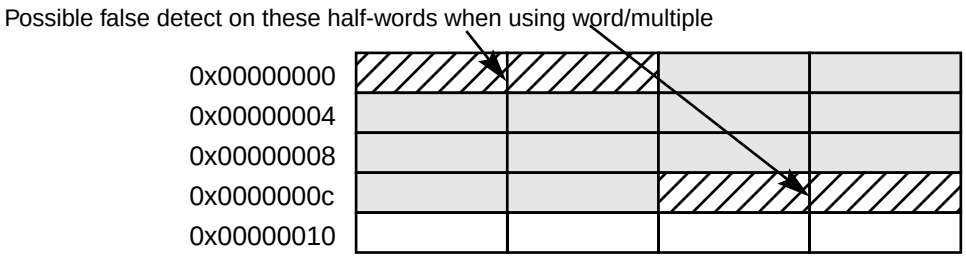


Figure 44-4. Partially Supported Watchpoints/Breakpoint Example

44.2.4.3 Context Dependent Filter

The core can be programmed to recognize only internal breakpoints when MSR[RI] = 1 (maskable mode) or to always recognize internal breakpoints (nonmaskable mode).

In maskable mode, when the core is programmed only to recognize internal breakpoints (when MSR[RI] =1), it is possible to debug all parts of the code, except when SRR0 and SRR1, DAR, and DSISR are busy as indicated by MSR[RI] = 0 (in the prologues and epilogues of exception handlers). Internal breakpoints detected when MSR[RI] = 0 are lost and debug counters do not count detected watchpoints. Detected watchpoints are always reported on the external pins, regardless of the value of MSR[RI].

In nonmaskable mode, when the core is programmed to recognize internal breakpoints, all parts of the code can be debugged. However, if an internal breakpoint is recognized when MSR[RI] = 0 (SRR0 and SRR1 are busy), the machine enters a nonrestartable state. See Section 6.1.5, “Recoverability after an Exception.”

The core defaults to maskable mode after reset. The core is put in nonmaskable mode by setting LCTRL2[BRKNOMSK], which controls all internal I- and L-breakpoints. See Section 44.5.1.5, “Load/Store Support AND-OR Control Register (LCTRL2).”

44.2.4.4 Ignore First Match

The ignore first match bit, ICTRL[IFM], facilitates the debugger's "continue" and "go from x" utilities for instruction breakpoints. When an instruction breakpoint is first enabled, the first instruction cannot cause an instruction breakpoint if ICTRL[IFM] = 1. This is used for "continue" utilities. If IFM = 0, every matched instruction causes an instruction breakpoint. This is used for "go from x". IFM is set by software and cleared by 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.

44.2.4.5 Generating Six Compare Types

The four compare types (equal, not equal, greater than, and less than) can be used to generate two additional compare types—greater than or equal to and less than or equal to. The greater-than-or-equal compare type can be generated by using the greater-than compare type and programming the comparator to the value in question minus 1. Likewise, the less-than-or-equal compare type can be generated by using the less-than compare type and programming the comparator to the value in question plus 1. This does not work for the following boundary cases:

- Less than or equal of the largest unsigned number (1111...1).
- Greater than or equal of the smallest unsigned number (0000...0).
- Less than or equal of the maximum positive number in signed mode (0111...1).
- Greater than or equal of the maximum negative number 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 44.5.1.5, "Load/Store Support AND-OR Control Register (LCTRL2)."

44.2.5 Load/Store Breakpoint Example

CMPE and CMPF are used for load/store addresses while CMPG and CMPH are used for load/store data.

The procedure is as follows:

1. Write the value in the appropriate comparator register, CMPE, CMPF, CMPG, or CMPH.
2. For load/store data, program the operand size in LCTRL1[CS*x*] and the byte mask in LCTRL1[C*x*BMSK].
3. Write the comparison type in LCTRL1[CT*x*]. For load/store data, program whether the operand is signed or unsigned LCTRL1[SUS*x*].
4. Select a watchpoint enable event:

- Define the load/store watchpoint event in LCTRL2[LW*x*LA] or LCTRL2[LW*x*LD]
- Enable the address or data event in LCTRL2[LW*x*LADC] or LCTRL2[LW*x*LDDC]

5. Disable instruction events affecting load/store watchpoints—Clear LW*x*IADC (LW*x*IA is a don't care).
6. Enable the watchpoint in LCTRL2[LW*x*EN].
7. Enable a trap on every watchpoint or every *N* watchpoints.

   Option: Enable trap on every load/store watchpoint in LCTRL2[SLW*x*EN] or on every *N* watchpoints in COUNT*x*. (Set CNTV to *n* and select the load/store watchpoint in CNTC).

8. Select whether breakpoints are maskable or nonmaskable in LCTRL2[BRKNOMSK].
9. Optionally select whether a load/store trap causes the debug mode to be entered in DER[LBRKE].

44.3 Development System Interface

It is often useful to debug a target system without making changes. However, sometimes it is impossible to add load to the lines connected to the existing system without disrupting its operation. The development system interface of the core enables debug of a target system with minimal cost and intrusiveness.

The development system interface of the core uses the development port, which is a dedicated serial port and, therefore, does not need any of the regular system interfaces.

The development port is a relatively inexpensive interface that allows a development system to operate in a lower frequency than the core's frequency and controls system activity when the core is in debug mode. It is also possible to debug the core using monitor debugger software, described in Section 44.4, "Software Monitor Debugger Support."

In debug mode the core fetches all instructions from the development port; data can be read from the development port and 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, two possible working modes are defined—debug mode enable and debug mode disable, described in Section 44.3.1.1, "Debug Mode Enable vs. Debug Mode Disable," are selected only during reset.

The user can work in debug mode directly out of reset or the core can be programmed to enter 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 combination with two levels of development port requests generated externally. Each of these can be programmed to be treated as a regular interrupt that causes the machine to branch to its

interrupt vector or as a special interrupt that causes debug mode entry. In debug mode, the **rfi** instruction returns the machine to its regular work mode. Figure 44-5 shows the relationship between debug mode logic and the rest of the core.

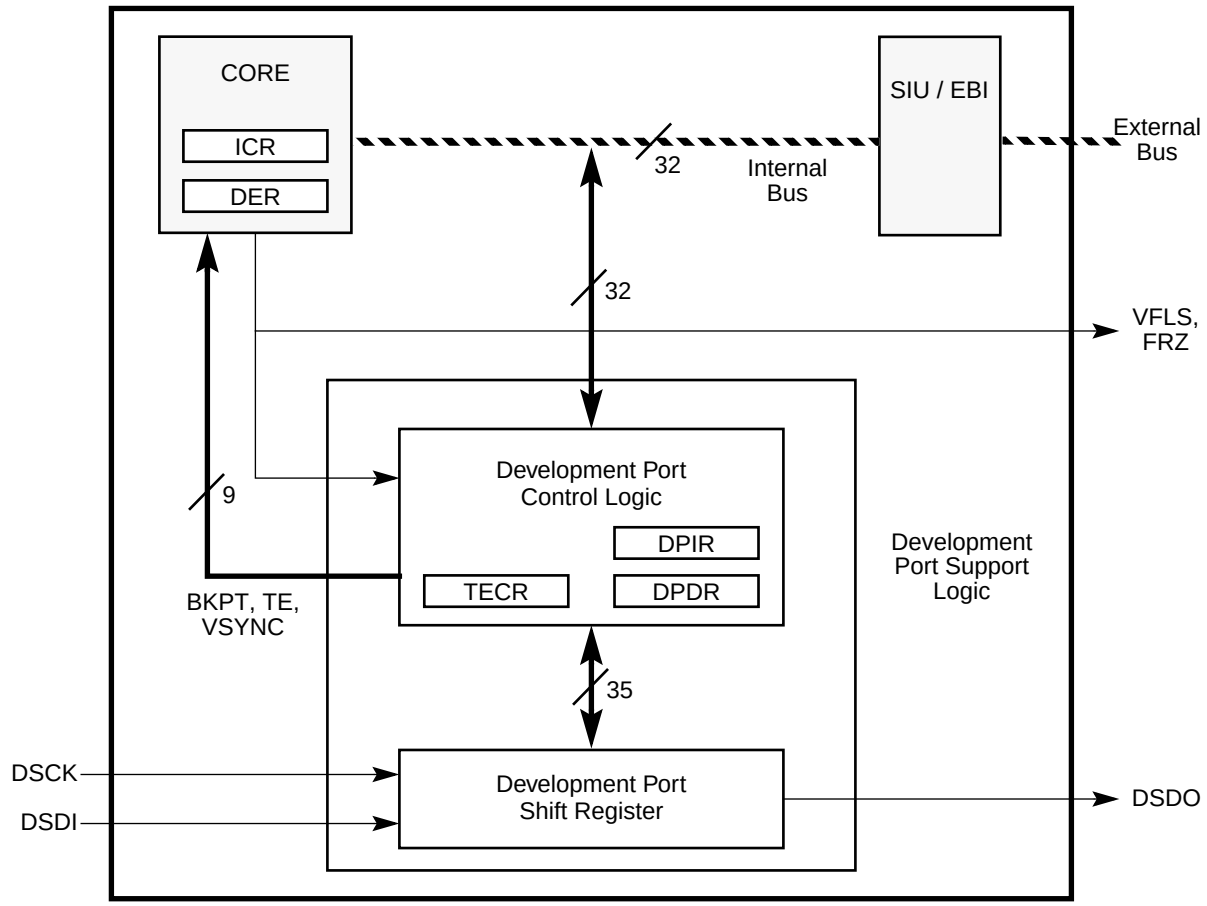


Figure 44-5. Functional Diagram of the MPC855T Debug Mode Support

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.

Trap enable mode shifts the following control signals into the core internal development support logic.

- Instruction trap enable bits for programming the instruction breakpoint dynamically.
- Load/store trap enable bits for programming the load/store breakpoint dynamically.
- Nonmaskable breakpoint is used to assert the nonmaskable external breakpoint.
- Maskable breakpoint is used to assert the maskable external breakpoint.
- 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. See **Section 44.3.2, "Development Port Communication**."

44.3.1 Debug Mode Operation

Figure 44-6 shows the debug mode logic implemented in the core.

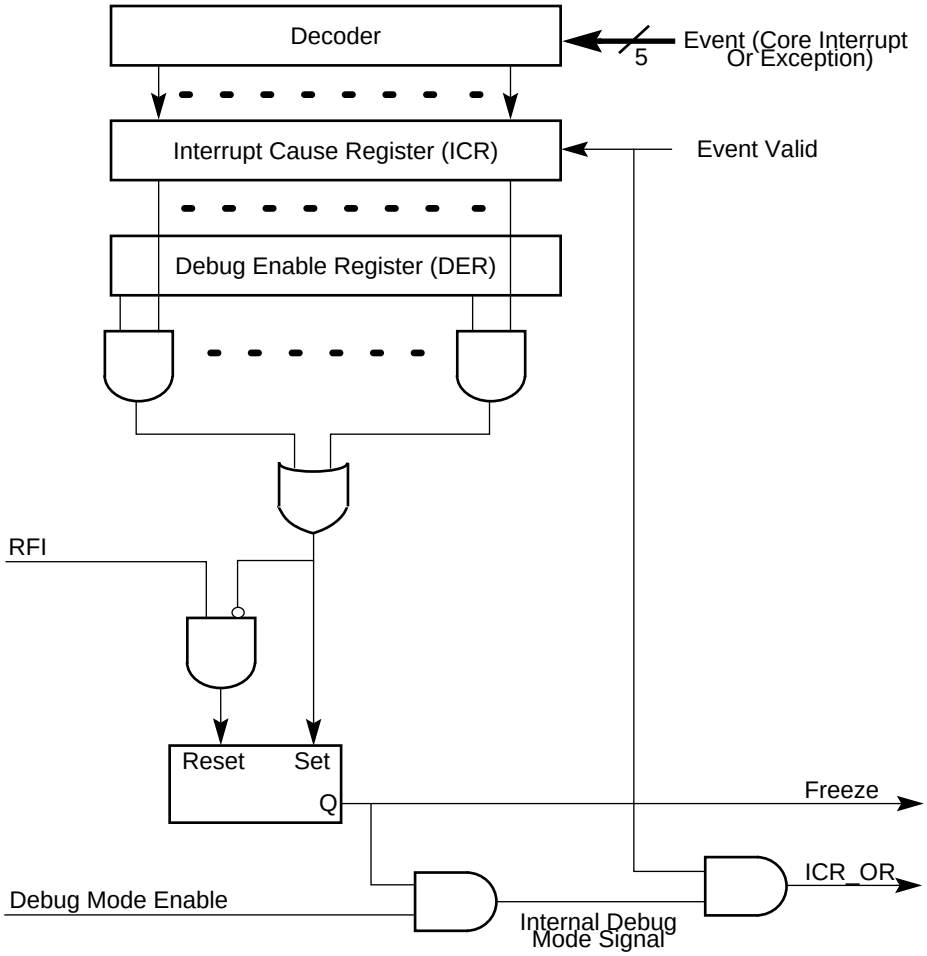


Figure 44-6. Debug Mode Logic Diagram

The debug mode of the core provides the development system with the following functions:

- Controls and maintains execution of the processor in all circumstances. The development port can force the core to enter debug mode even when external interrupts are disabled.
- Debug mode can be entered immediately out of reset, allowing the user to debug a system without using ROM.
- The debug enable register (DER) can be used to selectively enable events that cause the machine to enter debug mode.
- The interrupt cause register (ICR) indicates why debug mode is entered.
- After entry into debug mode, program execution continues from the where debug mode was entered.

- All instructions are fetched from the development port, while load/store accesses are performed on the real system memory in debug.
- 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 privileged state (MSR[PR] = 0) in debug mode, allowing execution of any instruction and access to any memory location.
- An OR signal of all interrupt cause register (ICR) 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 a nonexisting memory space.
- Caches and MMUs are frozen in debug mode. All accesses made during debug mode will be to the memory. Cache contents can only be accessed via SPRs.

44.3.1.1 Debug Mode Enable vs. Debug Mode Disable

For protection purposes, there are two working modes, debug mode enable and debug mode disable, which are selected once at reset. Debug mode is enabled by asserting DSCK during reset. The state of this pin is sampled three clocks before the negation of $\overline{\text{SRESET}}$. If DSCK is sampled negated, debug mode is disabled until a subsequent reset when DSCK is asserted. When debug mode is disabled, the internal watchpoint/breakpoint hardware remains operational and can be used for debugging by a software monitor program. Figure 44-7 is a timing diagram for the enabling debug mode.

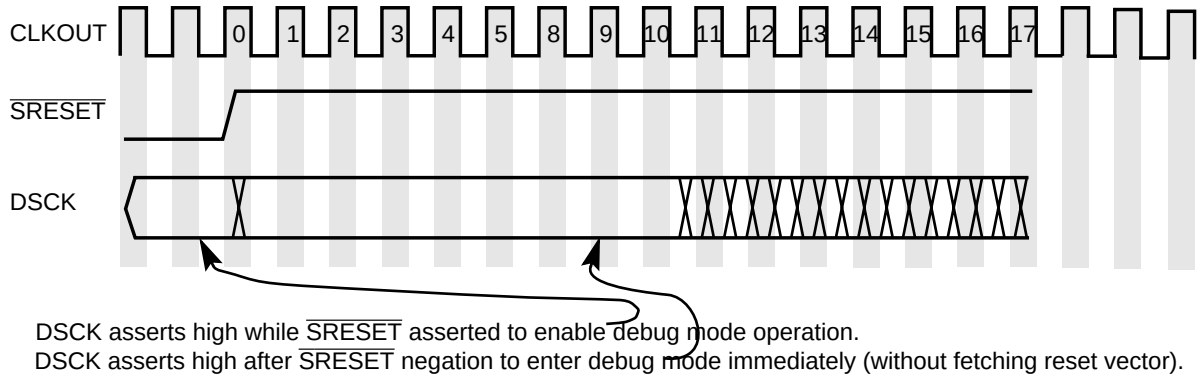


Figure 44-7. Debug Mode Reset Configuration Timing Diagram

Note that because $\overline{\text{SRESET}}$ negation time depends on an external pull-up resistor, any reference to $\overline{\text{SRESET}}$ negation time in this chapter refers to the time the MPC855T releases $\overline{\text{SRESET}}$. If $\overline{\text{SRESET}}$ rise time is long because of a large resistor, the setup time for 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 monitor debugger software. However, the processor never enters debug mode and the ICR and DER are used only for asserting and negating the freeze signal. For more information on the software monitor debugger support, see Section 44.4,

"Software Monitor Debugger Support." All development support registers accessible only when the core is in debug mode; therefore, the development system has full control of the core's development support features. For more information, see Table 44-15. If debug mode is enabled as described in this section, debug mode can be entered by the methods described in Section 44.3.1.2, "Entering Debug Mode."

44.3.1.2 Entering Debug Mode

By appropriately programming the development port during reset, debug mode can be entered immediately out of reset, thus allowing the user to debug a ROM-less system. If DSCK is asserted throughout $\overline{\text{SRESET}}$ assertion and then past $\overline{\text{SRESET}}$ negation, the processor takes a breakpoint exception and goes directly to debug mode instead of fetching the reset vector.

To avoid entering debug mode after reset, DSCK must be negated no later than seven clock cycles after $\overline{\text{SRESET}}$ negates, allowing the processor to jump to the reset vector and begin normal execution. If debug mode is entered immediately after reset, as shown in Figure 44-7, ICR[DPI] is set.

The user can enable events that can initiate debug mode and determine which events require regular interrupt handling.

The following 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 in the DER. The reset values of the enable bits allow use of the debug mode features without programming the DER in most cases. See Table 44-25.

- System reset, as a result of the assertion of $\overline{\text{SRESET}}$, as described in Section 6.1.2.1, "System Reset Interrupt (0x00100)"
- Checkstop, as described in Table 44-9
- Machine check interrupt
- Implementation-specific ITLB miss
- Implementation-specific ITLB error
- Implementation-specific DTLB miss
- Implementation-specific DTLB error
- External interrupt, recognized when MSR[EE] = 1
- Alignment exception
- Program exception
- Floating-point unavailable exception
- Decrementer interrupt, recognized when MSR[EE] = 1
- System call exception

- Trace, asserted when in single or branch trace mode, as described in Section 6.1.2.10, “Trace Exception (0x00D00)”
- Implementation-dependent software emulation exception
- Instruction breakpoint. Recognized only when MSR[RI] = 1, when breakpoints are maskable. Nonmaskable breakpoints are always recognized.
- Load/store breakpoint. Recognized only when MSR[RI] = 1, when breakpoints are maskable. Nonmaskable breakpoints are always recognized.
- Maskable breakpoint from the development port generated by external modules are recognized only when MSR[RI] = 1
- Development port nonmaskable interrupt resulting from a debug station request. Useful in some catastrophic events like an endless loop when MSR[RI] = 0. This may cause the machine to enter a nonrestartable state. See Section 6.1.5, “Recoverability after an Exception.”

The processor enters debug mode when at least one ICR bit is set, the corresponding DER bit 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 before fetching instructions from the development port. Section Chapter 6, “Exceptions,” gives the exact SRR0 and SRR1 values. If the core is in debug mode, the freeze indication is asserted, causing any properly programmed peripheral to stop. The development port should read the value of the ICR to get the cause of the debug mode entry. Reading the ICR clears all of its bits.

44.3.1.3 Debug Mode Indication

The fact that the core is in debug mode is broadcast to the external world using the value 0b11 on the VFLS pins. Debug mode indication will also be given on the FRZ pin. Note, however, that the FRZ indication can also be used by software monitors, as described in Section 44.4, “Software Monitor Debugger Support.”

44.3.1.4 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 DER[CKSTPE] is also set, the core enters debug mode rather then the checkstop state. Table 44-9 shows the various actions the core can take when a machine check interrupt is detected.

Table 44-9. Checkstop State and Debug Mode

MSR[ME]	Debug Mode Enable	DER[CHSTPE]	DER[MCIE]	Core Response to Machine Check Interrupt	ICR Value
0	0	X	X	Enter the checkstop state	0x20000000
1	0	X	X	Branch to machine check interrupt	0x10000000
0	1	0	X	Enter checkstop state	0x20000000

Table 44-9. Checkstop State and Debug Mode (continued)

MSR[ME]	Debug Mode Enable	DER[CHSTPE]	DER[MCIE]	Core Response to Machine Check Interrupt	ICR Value
0	1	1	X	Enter debug mode	0x20000000
1	1	X	0	Branch to machine check interrupt	0x10000000
1	1	X	1	Enter debug mode	0x10000000

44.3.1.5 Saving Machine State when Entering Debug Mode

If any load/store-type exception causes the store to enter debug mode, the critical information in DAR and DSISR must be saved before any other operation is performed. Failing to do so can cause information loss if the development software encounters another load/store-type exception. Because exceptions are treated differently in debug mode, there is no need to save SRR0 and SRR1.

44.3.1.6 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 an SPR and is accessed using **mtspr** and **mfspr** via special load/store cycles (see Table 44-14).

Exceptions are treated differently in debug mode; the ICR is updated on recognition of an exception according to the event that caused it. A special error indication (ICR_OR) is asserted for one clock cycle to notify the development port when an exception occurs. Execution continues in debug mode without changing SRR0 and SRR1. To allow the development system to detect the excepting instruction, ICR_OR is asserted before the next fetch. Not all exceptions are recognized in debug mode. Hardware does not generate breakpoints and watchpoints in debug mode, regardless of the value of MSR[RI]. On entering debug mode, MSR[EE] is cleared, forcing hardware to ignore external and decrementer interrupts.

Note that debug software must not set MSR[EE] in debug mode because the external interrupt event is a level signal. Because the core only reports and does not handle exceptions in debug mode, core hardware does not clear MSR[EE]. This event, if enabled, is recognized on every clock. When ICR_OR is asserted the development station should read the ICR to find what event caused the exception. Because SRR0 and SRR1 do not change, if an exception is recognized in debug mode, they change only once when entering debug mode. However, saving SRR0 and SRR1 when entering debug mode is unnecessary.

44.3.1.7 Exiting Debug Mode

The **rfi** instruction is used to exit from debug mode to return to the normal processor operation and to negate the freeze indication. The development system may monitor the

FRZ or FLS pins to make sure the MPC855T is out of debug mode. It is the responsibility of the debugger to read the ICR before performing the **rfi** instruction. Failing to do so forces the core to immediately reenter debug mode and to reassert the freeze indication if an asserted ICR bit has a corresponding enable bit set in the DER.

44.3.2 Development Port Communication

The development port provides a full duplex serial interface for communications between the internal development support logic and an external development tool. Figure 44-5 shows the relationship of the development support logic to the rest of the core. For clarity, the development port support logic is shown as a separate block.

44.3.2.1 Development Port Pins

The following development port pin functions are provided:

- Development serial clock
- Development serial data in
- Development serial data out
- Freeze

44.3.2.1.1 Development Serial Clock (DSCK)

DSCK is used at reset to enable debug mode, which can be entered either immediately following reset or for event-driven entry into debug mode as described in Section 44.3.1.2, “Entering Debug Mode.” The DSCK input must be driven either high or low at all times and must not be allowed to float. A typical target environment would pull this input low with a resistor. When the development port is in asynchronous clocked mode, the development serial clock (DSCK) is used to shift data into and out of the development port shift register. At the same time, the new msb of the shift register is presented at the DSDO pin.

The clock may be implemented as a free-running or gated clock. As discussed in Section 44.3.2.4, “Development Port Serial Communications–Trap Enable Mode,” and Section 44.3.2.5, “Development Port Serial Communications–Debug Mode,” data shifting is controlled by the ready and start signals, so the clock does not need to be gated with the serial transmissions.

44.3.2.1.2 Development Serial Data In (DSDI)

External logic presents data to be transferred into the development port shift register at the development serial data in pin (DSDI). When driven asynchronously with the system clock, data presented to DSDI must be stable at setup time before the rising edge of DSCK and at hold time after the rising edge of DSCK. When driven synchronously to the system clock, 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. DSDI is also used at reset to select the development port clock mode. See Section 44.3.2.3, “Development Port Serial Communications–Clock Mode.”

44.3.2.1.3 Development Serial Data Out (DSDO)

Debug mode logic uses the development serial data out pin (DSDO) to shift data out of the development port shift register. DSDO transitions are synchronous with DSCK or CLKOUT, depending on the clock mode.

44.3.2.1.4 Freeze

The freeze indication means that the processor is in debug mode (normal processor execution of user code is frozen). Freeze state is indicated on FRZ and is generated synchronously 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 FRZ, the freeze state is indicated by the value 0b11 on VFLS[0–1], shown in Figure 44-8.

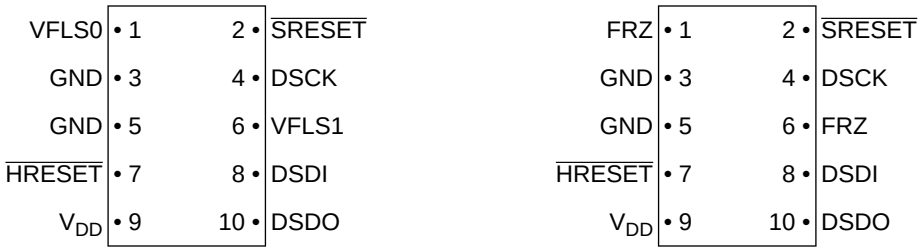


Figure 44-8. Development Port/BDM Connector Pinout Options

Internal freeze status can also be monitored through status in the data shifted out of the debug port.

44.3.2.2 Development Port Registers

The development port consists logically of three registers:

- The trap enable control register (TECR)
- The development port instruction register (DPIR)
- Development port data register (DPDR)

DPIR and DPDR are both implemented as the development port shift register, which also acts as a temporary holding register for data to be stored in the TECR.

44.3.2.2.1 Development Port Shift Register

Instructions and data are serially shifted into the 35-bit development port shift register from the DSDI. DSCK or CLKOUT is the shift clock, depending on the debug port clock mode. See Section 44.3.2.3, “Development Port Serial Communications–Clock Mode.”

The instructions or data are then transferred in parallel to the core and TECR. When the processor enters debug mode it fetches instructions from DPIR that cause an access to the development port shift register. These instructions are serially loaded into the shift register from 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 by executing **mfspr[DPDR]**. Data is also parallel loaded into the development port shift register from the core by executing **mtspr[DPDR]**. It is then serially shifted out to DSDO using DSCK or CLKOUT as the shift clock.

44.3.2.2.2 Trap Enable Control Register (TECR)

The TECR is a 9-bit register that is loaded from the development port shift register. The contents of TECR drives the six trap enable signals, two breakpoint signals, and VSYNC signal to the core. The transfer data to TECR commands send the appropriate bits to the TECR. The TECR 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 port shift register as the result of trap enable mode transmissions. The trap enable bits are reflected in ICTRL and LCTRL2. Section 44.5.1.1, “Comparator A–H Value Registers (CMPA–CMPH),” describes support registers.

44.3.2.2.3 Development Port Registers Decode

The development port shift register is selected when the core accesses DPIR or DPDR. Accesses to either register occur in debug mode and appears on the internal bus as an address and the assertion of an address attribute signal indicating that an SPR is being accessed. In debug mode, the core reads the DPIR to fetch all instructions; it reads and writes to the DPDR to transfer data between the core and external development tools. DPIR and DPDR are pseudo-registers; decoding either causes the development port shift register to be accessed. 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 results do not match, for example if an instruction is received when data is expected.

44.3.2.3 Development Port Serial Communications–Clock Mode

All development port serial transmissions are synchronous communications. The development port supports two ways to clock serial transmissions.

44.3.2.3.1 Asynchronous Clocked Mode—Using DSCK

The first clock mode is called asynchronous clocked since the input clock DSCK is asynchronous with CLKOUT. To ensure that data on DSDI is sampled correctly, transitions on DSDI must meet all setup and hold times with respect to the rising edge of DSCK. This clock mode allows communications with the port from a development tool which does not

have access to CLKOUT or where CLKOUT has been delayed or skewed. Figure 44-9 shows the serial communications asynchronous clocked timing.

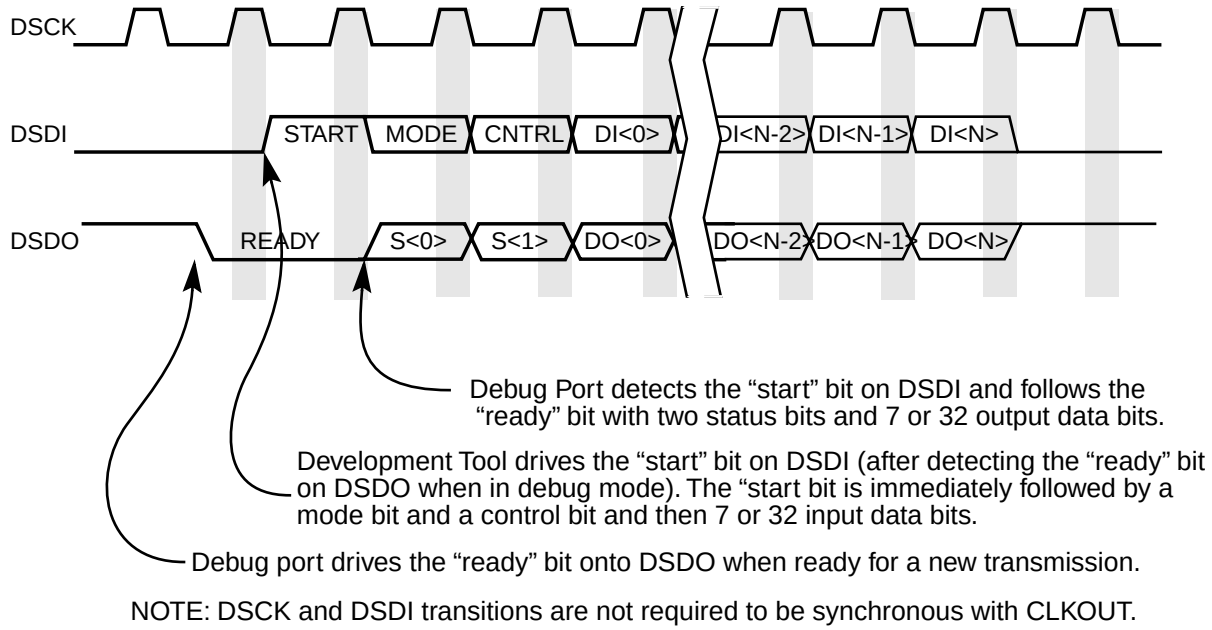


Figure 44-9. Asynchronous Clocked Serial Communications

44.3.2.3.2 Synchronous Self-Clocked Mode—Using CLKOUT

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 CLKOUT rising edge. The data rate for this mode is always the same as the system clock. The timing diagram in Figure 44-10 shows the serial communications synchronous self-clocked timing.

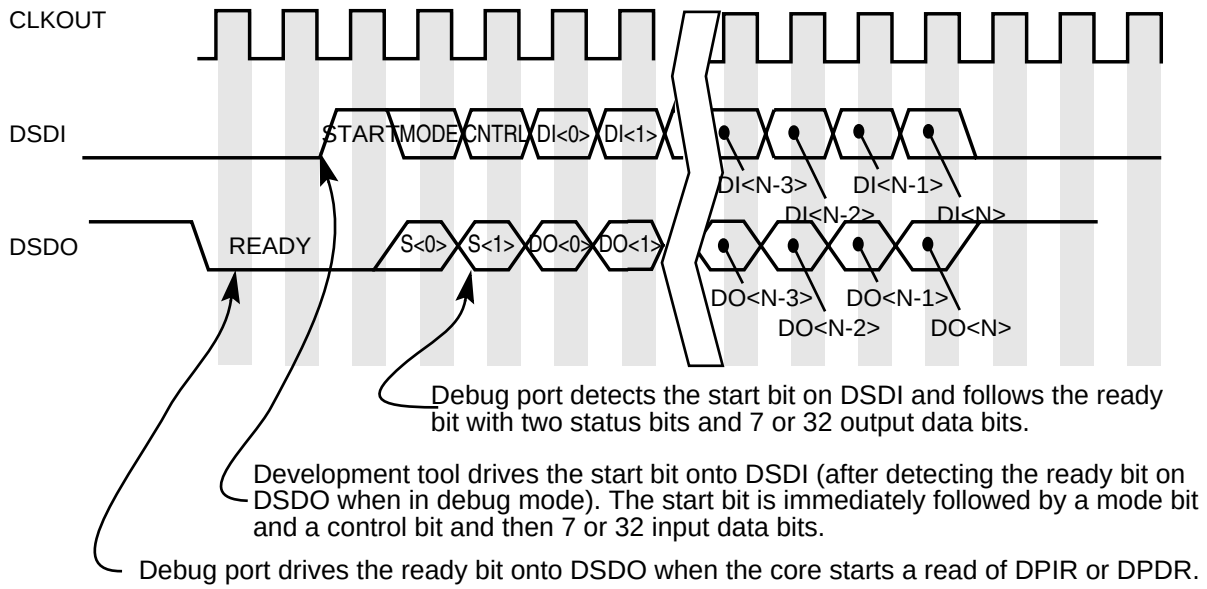


Figure 44-10. Synchronous Self-Clocked Serial Communications

44.3.2.3.3 Selection of Development Port Clock Mode

The selection of clocked or self-clocked mode is made at reset. The state of the DSDI input is latched eight clocks after negation of $\overline{\text{SRESET}}$. 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 44-11 shows the clock mode selection after reset.

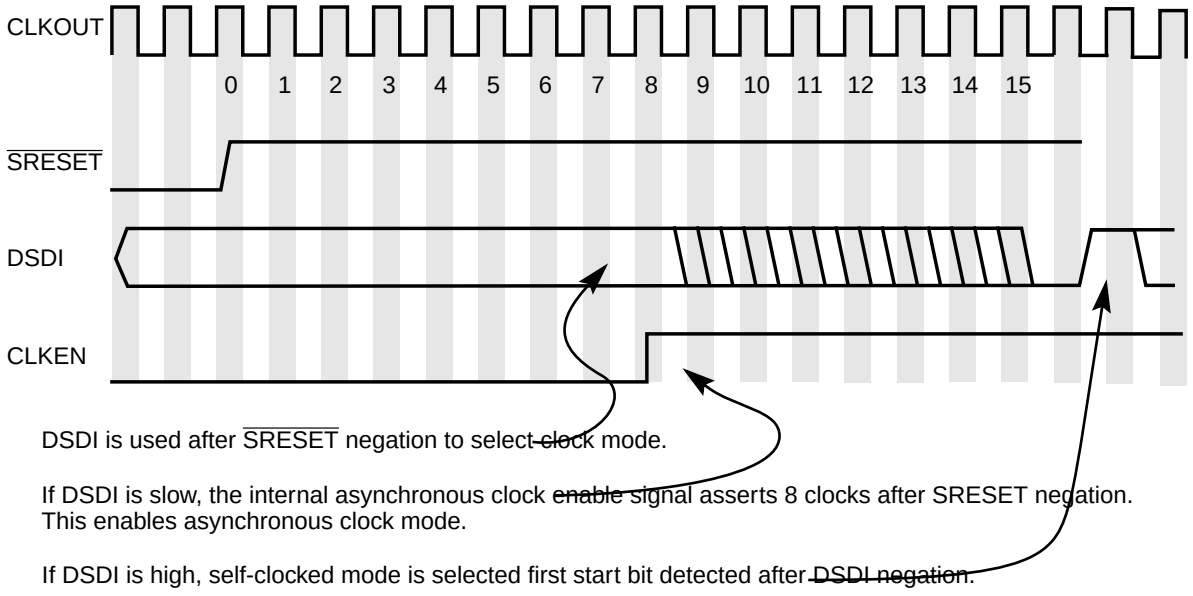


Figure 44-11. Enabling Clock Mode after Reset

Since DSDI is used to select the development port clock scheme, it is necessary to prevent any transitions on DSDI during clock mode select. The port will not begin scanning for the start bit of a serial transmission until 16 clocks after the negation of $\overline{\text{SRESET}}$. If DSDI is

asserted 16 clocks after $\overline{\text{SRESET}}$ negates, the port waits until DSDI is negated to begin scanning for the start bit.

44.3.2.4 Development Port Serial Communications–Trap Enable Mode

When not in debug mode, the development port begins communicating by setting DSDO (the msb of the 35-bit development port shift register) low to indicate 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 transmission or a 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 trap enable and VSYNC bits of the TECR or into the breakpoints bits of the TECR.

44.3.2.4.1 Serial Data Into Development Port

The development port shift register is 35 bits wide, but trap enable mode transmissions only use 10 of the 35 bits as the following—the start/ready bit, a mode/status bit, a control/status bit, and the 7 least-significant data bits.

Table 44-10 shows the encoding of trap data shifted into the (through DSDI).

Table 44-10. Trap Enable Data Shifted into Development Port Shift Register

Start	Mode	Control	1st	2nd	3rd	4th	1st	2nd	VSYNC	Function
			Instruction				Data			
			Watchpoint Trap Enables							
1	1	0	0 = Disabled 1 = Enabled							Transfer data to trap enable control register

Table 44-11 shows the encoding of debug port command data shifted into the development port shift register.

Table 44-11. Debug Port Command Shifted Into Development Port Shift Register

Start	Mode	Control	Extended Opcode		Major Opcode	Function
1	1	1	x	x	00000	NOP
					00001	$\overline{\text{HRESET}}$ request
					00010	$\overline{\text{SRESET}}$ 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 enables and VSYNC functions are described in Section 44.2, “Watchpoints and Breakpoints Support,” and Section 44.1, “Tracking Program Flow.” The debug port command function allows the development tool to either assert or negate breakpoint requests, reset the processor, activate or deactivate the fast download procedure.

44.3.2.4.2 Serial Data Out of Development Port

In trap enable mode there is no data from the core out of the development port. Data out of the development port in the trap enable mode is shown in Table 44-12.

Table 44-12. Status/Data Shifted Out of Development Port Shift Register

Ready	Status [0–1]		Data			Function
			Bit 0	Bit 1	Bits 2–31 or 2–6 Depending on Input Mode	
(0)	0	0	Data			Valid data from core
(0)	0	1	Freeze status [1]	Download procedure in progress [2]	1s	Sequencing error
(0)	1	0			1s	Core interrupt
(0)	1	1			1s	Null

1 The freeze status is 1 when the core is in debug mode. Otherwise it is 0.
2 The “Download Procedure In Progress” status is asserted (0) when the debug port in the download procedure is negated. Otherwise it is set to 1.

The “Valid Data from Core” and “Core Interrupt” functions cannot occur in trap enable mode. When not in 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 port ignores the data shifted in while the sequencing error is shifting out and is treated as a no-op function. The null output encoding is used to indicate that the previous transmission had no associated errors. When not in debug mode, ready is asserted at the end of each transmission. If debug mode is not enabled and transmission errors can be guaranteed not to occur, the status output is not needed.

44.3.2.5 Development Port Serial Communications–Debug Mode

Debug mode is a superset of trap enable mode. All of the trap enable mode functionality is available, with the following additions.

- In debug mode, the development port starts communications by setting DSDO low to indicate that the core is trying to read an instruction from DPIR or data from DPDR.
- When the core writes data to the port to be shifted out, the ready bit is not set. Instead, the port waits for the core to read the next instruction before asserting ready. This allows duplex operation of the serial port and lets the port control all transmissions from the external development tool. After detecting this ready status the external development tool begins transmitting to the development port with a start bit (logic high) on DSDI.

44.3.2.5.1 Serial Data Into Development Port

In debug mode the 35 bits of the development 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 sent with the mode bit cleared indicating a 32-bit data field. Table 44-13 shows the encoding of data shifted into the development port shift register through DSDI. Data values in the last two functions other than those specified are reserved.

Table 44-13. Debug Instructions/Data Shifted Into Development Port Shift Register

Start	Mode	Control	Instruction/Data (32 Bits)		Function
			Bits 0–6	Bits 7–31	
1	0	0	Core instruction		Transfer instruction to core
1	0	1	Core data		Transfer data to core
1	1	0	Trap enable bits	Not exist	Transfer data to trap enable control register
1	1	1	0b001_1111	Not exist	Negate breakpoint requests to core
1	1	1	0	Not exist	NOP

Note: See Table 44-10 for details on trap enable bits.

Transmissions from the debug port on DSDO begin with a zero or ready bit, indicating that the core is trying to read an instruction or data from the port. The external development tool

must wait until it sees DSDO go low before sending the next transmission. The control bit distinguishes instructions from data, allowing the development port to detect that an instruction 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 port to transfer data to the trap enable control register. The debug port command function allows the development tool to either negate breakpoint requests, reset the processor, activate, or deactivate the fast download procedure. The NOP function provides a null operation for use when there is data or a response to be shifted out of the data register. The appropriate next instruction or command will be determined by the value of the response or data shifted out.

44.3.2.5.2 Serial Data Out of Development Port

The encoding of data shifted out of the development port shift register in debug mode is the same as for trap enable mode, as shown in Table 44-12. The valid data encoding is used when data has been transferred from the core to the development port shift register as the result of an instruction to move the contents of a GPR to the DPDR. The valid data encoding has the highest priority of all status outputs and is reported even if an interrupt occurs at the same time. Because a sequencing error cannot occur when data is valid, there is no priority conflict with the sequencing error status. Also, an interrupt recognized when there is valid data is not related to the execution of an instruction, therefore, a valid data status is output and the interrupt status is saved for the next transmission.

The sequencing error encoding indicates that the inputs from the external development tool are not what the development port and/or the core was expecting. There are two possible causes for this error:

- The processor was trying to read instructions and data was shifted into the development port.
- The processor was trying to read data and an instruction was shifted into the development 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 occurred. Because 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 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 a new instruction, trap enable, or command.

The interrupt occurred encoding indicates that the core encountered an interrupt during the execution of the previous instruction in debug mode. Interrupts may 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 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 was transferred from the core to the development port shift register.

44.3.2.5.3 Fast Download Procedure

The fast download procedure downloads a block of data from the debug tool into the system memory by repeating the sequence of transactions shown in Figure 44-12 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 44-12. Download Procedure Code Example

In this example, RX = r31 and RY = r30. The sequence is repeated until the end download procedure command is issued to the debug port. GPR31 temporarily stores the data value. Before issuing the start download procedure command, the value of the first memory block address -4 must be written into GPR30. To end the download, an end download procedure command should be issued to the debug port and an additional data transaction should be sent by the development tool. This data word is not placed into system memory, but it is needed to stop the procedure.

For large blocks of data this sequence may take a long time to complete. The fast download procedure can reduce this time by eliminating the need to transfer instructions in the loop to the debug port. The only transactions needed are those that transfer the data to be placed in the system memory. Figure 44-13 shows the time benefit of the fast download procedure.

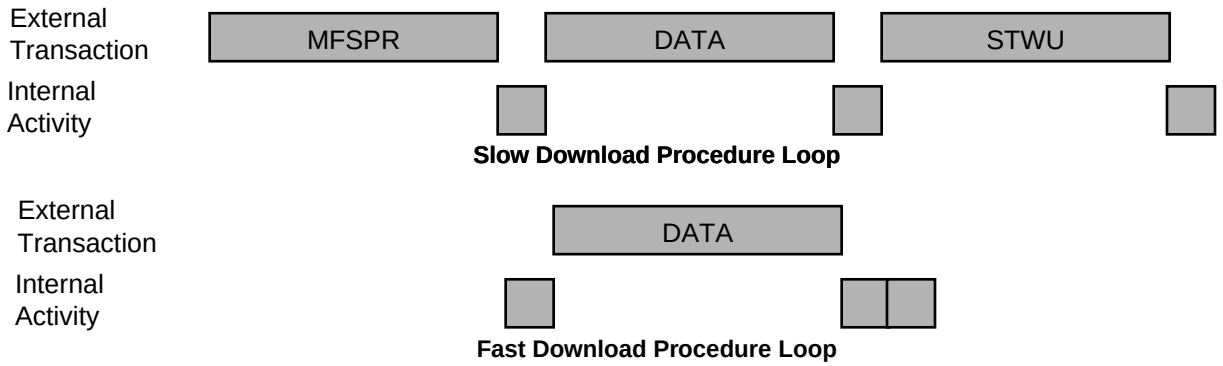


Figure 44-13. Fast and Slow Download Procedure Loops

44.4 Software Monitor Debugger Support

With debug mode disabled, a software monitor debugger can use the development support features defined in the core; all events result in regular exception handling (the processor resumes execution in the corresponding interrupt handler). The ICR and DER influence only the assertion and negation of the freeze signal.

44.4.1 Freeze Indication

The internal freeze signal connects to all relevant internal modules, which can be programmed to stop all operations in response to the assertion of the freeze signal. To enable a software monitor debugger to broadcast the fact that debug software has executed, it is possible to assert and negate the internal freeze signal when debug mode is disabled.

The assertion of the freeze signal is broadcast externally over FRZ. As shown in Figure 44-6, the ICR and DER control assertion and negation of the freeze signal when debug mode is disabled. To assert the freeze signal, software must program the relevant DER bits, but to negate the freeze line, the software must read the ICR to clear it and execute an **rfi**. If the ICR is not cleared before the **rfi** is executed, the freeze signal is not negated. Therefore, it is possible to nest inside a software monitor debugger without affecting the value of the freeze line, even though **rfi** may execute a few times. Only before the last **rfi** instruction does software need to clear the ICR. The above mechanism allows software to accurately control the assertion and negation of the freeze line.

44.5 Development Support Programming Model

The MPC855T implements a set of SPRs, shown in Table 44-14, that support debugging and reside in the control registers space. They can be accessed using mtspr and mfspr.

Table 44-14. MPC855T-Specific Development Support and Debug SPRs

SPR Number			Name	Serialization Performed in Response to Access
Decimal	SPR [5–9]	SPR [0–4]		
Development Support Registers				
144	00100	10000	CMPA	Fetch sync on write
145	00100	10001	CMPB	Fetch sync on write
146	00100	10010	CMPC	Fetch sync on write
147	00100	10011	CMPD	Fetch sync on write
150	00100	10110	COUNTA	Fetch sync on write
151	00100	10111	COUNTB	Fetch sync on write
152	00100	11000	CMPE	Write: Fetch sync Read: Sync relative to load/store operations

Table 44-14. MPC855T-Specific Development Support and Debug SPRs (continued)

SPR Number			Name	Serialization Performed in Response to Access
Decimal	SPR [5–9]	SPR [0–4]		
153	00100	11001	CMPF	Write: Fetch sync Read: Sync relative to load/store operations
154	00100	11010	CMPG	Write: Fetch sync Read: Sync relative to load/store operations
155	00100	11011	CMPH	Write: Fetch sync Read: Sync relative to load/store operations
156	00100	11100	LCTRL1	Write: Fetch sync Read: Sync relative to load/store operations
157	00100	11101	LCTRL2	Write: Fetch sync Read: Sync relative to load/store operations
158	00100	11110	ICTRL	Fetch sync on write
159	00100	11111	BAR	Write: Fetch sync Read: Sync relative to load/store operations. See Section 4.1.2.1, “DAR, DSISR, and BAR Operation.”
Debug Registers				
148	00100	10100	ICR	Fetch sync on write
149	00100	10101	DER	Fetch sync on write
630	10011	10110	DPDR	Read and Write

The development support/debug registers are protected as described in Table 44-15. Note the special behavior of the ICR and DPDR.

Table 44-15. Development Support/Debug Registers Protection

Operation	MSR[PR]	Debug Mode Enable	In Debug Mode	Result
Read register	0	0	X	Read is performed. (When reading ICR, it is also cleared.)
	0	1	0	Read is performed. (When reading ICR, it is not cleared.)
	0	1	1	Read is performed. (When reading ICR, it is also cleared.)
	1	X	X	Read is not performed, program interrupt is generated. (When reading ICR, it is not cleared.)
Write register	0	0	X	Write is performed. (Write to ICR or DPDR is ignored, the register is not modified and no interrupt is generated.)
	0	1	0	Write is ignored.
	0	1	1	Write is performed. (Write to ICR is ignored, the register is not modified and no interrupt is generated.)
	1	X	X	Write is not performed, program interrupt is generated.

44.5.1 Development Support Registers

The following sections describe the development support registers.

44.5.1.1 Comparator A–H Value Registers (CMPA–CMPH)

The comparator value registers (CMPA–CMPH) hold the instruction and data to be used in comparisons. Figure 44-14 shows CMPA–CMPD, which are used for instruction address bus comparisons. Because instructions are 32 bits wide (word), bits 30–31 are not used.

Bit	0	1	2	3	4	5	6	…	29	30	31
Field	CMPV									—	
Reset	Undefined										
R/W	R/W										
SPR	144 (CMPA), 145 (CMPB), 146 (CMPC), 147 (CMPD)										

Figure 44-14. Comparator A–D Value Register (CMPA–CMPD)

Table 44-16 describes CMPA–CMPD fields.

Table 44-16. CMPA–CMPD Field Descriptions

Bits	Name	Description
0–29	CMPV	Address bits to be compared.
30–31	—	Reserved.

Figure 44-15 shows CMPE–CMPF, which are used for load/store address bus comparisons.

Bit	0	1	2	3	4	5	6	7	8	9	…	31
Field	CMPV											
Reset	Undefined											
R/W	R/W											
SPR	152 (CMPE),153 (CMPF)											

Figure 44-15. Comparator E–F Value Registers (CMPE–CMPF)

Table 44-17 describes CMPE–CMPF fields.

Table 44-17. CMPE–CMPF Field Descriptions

Bits	Name	Description
0–31	CMPV	Address bits to be compared.

Figure 44-15 shows CMPG–CMPH, which are used for load/store data bus comparisons.

Bit	0	1	2	3	4	5	6	7	8	9	…	31
Field	CMPV											
Reset	Undefined											
R/W	R/W											
SPR	154 (CMPG),155 (CMPH)											

Figure 44-16. Comparator G–H Value Registers (CMPG–CMPH)

Table 44-18 describes CMPG–CMPH fields.

Table 44-18. CMPG–CMPH Field Descriptions

Bits	Name	Description
0–31	CMPV	Data bits to be compared.

44.5.1.2 Breakpoint Address Register (BAR)

The breakpoint address register (BAR), shown in Figure 44-17, is used to hold the address of the load/store cycle that generated a breakpoint.

Bit	0	1	2	3	4	5	6	7	8	9	…	31
Field	BARV											
Reset	Undefined											
R/W	R/W											
SPR	159											

Figure 44-17. Breakpoint Address Register (BAR)

Table 44-19 describes BAR fields,

Table 44-19. BAR Field Descriptions

Bits	Name	Description
0–31	BARV	The address of the load/store cycle that generated the breakpoint.

44.5.1.3 Instruction Support Control Register (ICTRL)

The instruction support control register (ICTRL), shown in Figure 44-18, is used to configure instruction breakpoint operations.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	CTA			CTB			CTC			CTD			IW0		IW1	
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	0000_0000_0000_0000															
R/W	R/W															
SPR	158															

Figure 44-18. Instruction Support Control Register (ICTRL)

Table 44-20 describes ICTRL fields.

Table 44-20. ICTRL Field Descriptions

Bits	Name	Description
0–2	CTA	Compare type of comparator A–D 0xx Not active (reset value) 100 Equal 101 Less than 110 Greater than 111 Not equal
3–5	CTB	
6–8	CTC	
9–11	CTD	
12–13	IW0	Instruction first watchpoint programming. 0x Not active (reset value) 10 Match from comparator A 11 Match from comparators (A & B)
14–15	IW1	Instruction second watchpoint programming. 0x Not active (reset value) 10 Match from comparator B 11 Match from comparators (A | B)
16–17	IW2	Instruction third watchpoint programming. 0x Not active (reset value) 10 Match from comparator C 11 Match from comparators (C & D)
18–19	IW3	Instruction fourth watchpoint programming. 0x Not active (reset value) 10 Match from comparator D 11 Match from comparators (C | D)
20	SIW0EN	Software trap enable selection of instruction watchpoints 0–3. 0 Trap disabled (reset value) 1 Trap enabled
21	SIW1EN	
22	SIW2EN	
23	SIW3EN	

Table 44-20. ICTRL Field Descriptions (continued)

Bits	Name	Description
24	DIW0EN	Development port trap enable selection of the instruction watchpoints 0–3 (read-only bit). 0 Trap disabled (reset value) 1 Trap enabled
25	DIW1EN	
26	DIW2EN	
27	DIW3EN	
28	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”).
29–31	ISCT_SER	Instruction fetch show cycle/core serialize control. Changing the instruction show cycle programming takes effect only from the second instruction after the **mtspr[ICTRL]**. 000 Core is fully serialized; show cycle is performed for all fetched instructions (reset value). 001 Core is fully serialized; show cycle is performed for all changes in program flow. 010 Core is fully serialized; show cycle is performed for all indirect changes in program flow. 011 Core is fully serialized; no show cycles is performed for fetched instructions. 100 Illegal. 101 Core is not serialized (normal mode); show cycle is performed for all changes in the program flow. If the fetch of the target of a direct branch is aborted by the core (because of an exception), the target is not always visible on the external pins. Program trace is not affected by this phenomenon. 110 Core is not serialized (normal mode; show cycle is performed for all indirect changes in program flow. 111 Core is not serialized (normal mode); no show cycle is performed for fetched instructions. When ISCT_SER = 010 or 110, the $\overline{\text{STS}}$ functionality of OP2/MODCK1/$\overline{\text{STS}}$ must be enabled by writing 10 or 11 to SIUMCR[DBGC]. The address on the external bus should be sampled only when $\overline{\text{STS}}$ is asserted.

44.5.1.4 Load/Store Support Comparators Control Register (LCTRL1)

The load/store support comparators control register (LCTRL1), shown in Figure 44-19, is used to configure load/store address breakpoint operations.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	CTE			CTF			CTG			CTH			CRWE		CRWF	
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	CSG		CSH		SUSG	SUSH	CGBMSK				CHBMSK				—	
Reset	0000_0000_0000_0000															
R/W	R/W															
SPR	156															

Figure 44-19. Load/Store Support Comparators Control Register (LCTRL1)

Table 44-21 describes LCTRL1 fields.

Table 44-21. LCTRL1 Field Descriptions

Bits	Name	Description
0–2	CTE	Compare type, comparators E–H. 0xx Not active (reset value) 100 Equal 101 Less than 110 Greater than 111 Not equal
3–5	CTF	
6–8	CTG	
9–11	CTH	
12–13	CRWE	Select match on read/write of comparators E and F. 0x Don't care (reset value) 10 Match on read 11 Match on write
14–15	CRWF	
16–17	CSG	Compare size, comparator G and H. 00 Reserved 01 Word 10 Half-word 11 Byte
18–19	CSH	
20	SUSG	Signed/unsigned operating mode for comparator G and H. 0 Unsigned 1 Signed
21	SUSH	
22–25	CGBMSK	Byte mask for comparator G and H. 0000 All bytes are not masked 0001 Last byte of the word is masked … 1111 All bytes are masked
26–29	CHBMSK	
30–31	—	Reserved

44.5.1.5 Load/Store Support AND-OR Control Register (LCTRL2)

The load/store support AND-OR control register (LCTRL2), shown in Figure 44-21, is used to configure load/store watchpoint operations.

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	
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	LW1LADC	LW1LD		LW1LDDC	BRKNOMSK	—							DLW0EN	DLW1EN	SLW0EN	SLW1EN
Reset	0000_0000_0000_0000															
R/W	R/W															
SPR	157															

Figure 44-20. Load/Store Support AND-OR Control Register (LCTRL2)

Table 44-22 describes LCTRL2 fields.

Table 44-22. LCTRL2 Field Descriptions

Bits	Name	Description
0	LW0EN	First load/store watchpoint enable bit. 0 Watchpoint not enabled (reset value) 1 Watchpoint enabled
1–2	LW0IA	First load/store watchpoint instruction watchpoint selection. 00 First instruction watchpoint 01 Second Instruction watchpoint 10 Third instruction watchpoint 11 Fourth Instruction watchpoint
3	LW0IADC	First load/store watchpoint care/don't care instruction events. 0 Don't care 1 Care
4–5	LW0LA	First load/store watchpoint load/store 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)
6	LW0LADC	First load/store watchpoint care/don't care load/store address events. 0 Don't care 1 Care
7–8	LW0LD	First load/store watchpoint load/store 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)
9	LW0LDDC	First load/store watchpoint care/don't care load/store data events. 0 Don't care 1 Care
10	LW1EN	Second load/store watchpoint enable bit. 0 Watchpoint not enabled (reset value) 1 Watchpoint enabled
11–12	LW1IA	Second load/store watchpoint load/store address watchpoint selection. 00 First instruction watchpoint 01 Second instruction watchpoint 10 Third instruction watchpoint 11 Fourth instruction watchpoint
13	LW1IADC	Second load/store watchpoint care/don't care load/store address events. 0 Don't care 1 Care
14–15	LW1LA	Second load/store watchpoint load/store 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)
16	LW1LADC	Second load/store watchpoint care/don't care load/store address events. 0 Don't care 1 Care

Table 44-22. LCTRL2 Field Descriptions (continued)

Bits	Name	Description
17–18	LW1LD	Second load/store watchpoint load/store 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)
19	LW1LDDC	Second load/store watchpoint care/don't care load/store data events. 0 Don't care 1 Care
20	BRKNOMSK	Internal breakpoints nonmask bit (controls both instruction and load/store breakpoints). 0 Masked mode, breakpoints are recognized only when MSR[RI] =1 (reset value). 1 Nonmasked mode, breakpoints are always recognized.
21–27	—	Reserved
28	DLW0EN	Development port trap enable selection of the first load/store watchpoint (read-only bit). 0 Trap disabled (reset value) 1 Trap enabled
29	DLW1EN	Development port trap enable selection of the second load/store watchpoint (read-only bit).
30	SLW0EN	Software trap enable selection of the first load/store watchpoint. 0 Trap disabled (reset value) 1 Trap enabled
31	SLW1EN	Software trap enable selection of the second load/store watchpoint. 0 Trap disabled (reset value) 1 Trap enabled

Programming each watchpoint consists of three control register fields—LWxIA, LWxLA, and LWxLD. All three conditions must be detected to assert a watchpoint.

44.5.1.6 Breakpoint Counter Value and Control Registers (COUNTA/COUNTB)

The breakpoint counter value and control registers (COUNTA/COUNTB), shown in Figure 44-21, can be programmed with the preset count value and counter source.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	CNTCV															
Reset	Undefined															
R/W	R/W															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—														CNTC	
Reset	0000_0000_0000_0000															
R/W	R/W															
SPR	150 (COUNTA), 151 (COUNTB)															

Figure 44-21. Breakpoint Counter Value and Control Registers (COUNTA/COUNTB)

Table 44-23 describes COUNTA and COUNTB fields.

Table 44-23. COUNTA/COUNTB Field Descriptions

Bits	Name	Description
0–15	CNTV	Counter preset value
16–29	—	Reserved
30–31	CNTC	Counter source select 00 Not active (reset value) 01 Instruction first (COUNTA)/second (COUNTB) watchpoint 10 Load/store first (COUNTA)/second (COUNTB) watchpoint 11 Reserved

44.5.2 Debug Mode Registers

The debug registers are described in the following sections.

44.5.2.1 Interrupt Cause Register (ICR)

The ICR indicates the reason that debug mode was entered. ICR bits are set by the hardware and cleared when the register is read. Attempts to write to ICR are ignored. All bits are cleared when exiting reset.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—	RST	CHSTP	MCI	—		EXTI	ALI	PRI	FPUVI	DECI	—		SYSI	TR	—
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—	SEI	ITLBMS	DTLBMS	ITLBER	DTLBER	—						LBRK	IBRK	EBRK	DPI
Reset	0000_0000_0000_0000															
R/W	R/W															
SPR	148															

Figure 44-22. Interrupt Cause Register (ICR)

ICR protection is described in Table 44-15. Table 44-24 describes ICR fields.

Table 44-24. ICR Field Descriptions

Bits	Name	Description
0	—	Reserved
1	RST	Reset interrupt bit. Set when the $\overline{\text{SRESET}}$ is asserted.
2	CHSTP	Check stop bit. Set when the machine check interrupt is asserted and MSR[ME] = 0. Results in debug mode entry if debug mode is enabled and the corresponding enable bit is set. Otherwise, the processor enters checkstop state.
3	MCI	Machine check interrupt bit. Set when the machine check interrupt is asserted and MSR[ME] =1. Causes debug mode entry if debug mode is enabled and the corresponding enable bit is set.
4–5	—	Reserved
6	EXTI	External interrupt bit. Set when the external interrupt is asserted. Causes debug mode entry if debug mode is enabled and the corresponding enable bit is set.
7	ALI	Alignment interrupt bit. Set when the alignment interrupt is asserted. Causes debug mode entry if debug mode is enabled and the corresponding enable bit is set.
8	PRI	Program interrupt bit. Set when the program interrupt is asserted. Causes debug mode entry if debug mode is enabled and the corresponding enable bit is set.
9	FPUVI	Floating-point unavailable interrupt bit. Set when the floating-point unavailable interrupt is asserted. Causes debug mode entry if debug mode is enabled and the corresponding enable bit is set.
10	DECI	Decrementer interrupt bit. Set when the decrementer interrupt is asserted. Causes debug mode entry if debug mode is enabled and the corresponding enable bit is set.
11–12	—	Reserved
13	SYSI	System call interrupt bit. Set when the system call interrupt is asserted. Causes debug mode entry if debug mode is enabled and the corresponding enable bit is set.
14	TR	Trace interrupt bit. Set when in single-step mode or when in branch trace mode. Causes debug mode entry if debug mode is enabled and the corresponding enable bit is set.
15–16	—	Reserved
17	SEI	Implementation-dependent software emulation interrupt. Set when the floating-point assist interrupt is asserted. Causes debug mode entry if debug mode is enabled and the corresponding enable bit is set.

Table 44-24. ICR Field Descriptions (continued)

Bits	Name	Description
18	ITLBMS	Implementation-specific ITLB miss. Set as a result of an ITLB miss. Causes debug mode entry if debug mode is enabled and the corresponding enable bit is set.
19	DTLBMS	Implementation-specific DTLB miss. Set as a result of an DTLB miss. Causes debug mode entry if debug mode is enabled and the corresponding enable bit is set.
20	ITLBER	Implementation-specific ITLB error. Set as a result of an ITLB error. Causes debug mode entry if debug mode is enabled and the corresponding enable bit is set.
21	DTLBER	Implementation-specific DTLB error. Set as a result of an DTLB error. results in debug mode entry if debug mode is enabled and the corresponding enable bit is set.
22–27	—	Reserved
28	LBRK	Load/store breakpoint interrupt bit. Set as a result of the assertion of an load/store breakpoint. Causes debug mode entry if debug mode is enabled and the corresponding enable bit is set.
29	IBRK	Instruction breakpoint interrupt bit. Set as a result of the assertion of an instruction breakpoint. Causes debug mode entry if debug mode is enabled and the corresponding enable bit is set.
30	EBRK	External breakpoint interrupt bit (development port, internal or external modules). Set as a result of the assertion of an external breakpoint. Causes debug mode entry if debug mode is enabled and the corresponding enable bit is set.
31	DPI	Development port interrupt bit. Set by the development port as a result of a debug station nonmaskable request or when entering debug mode immediately out of reset. Causes debug mode entry if debug mode is enabled and the corresponding enable bit is set.

44.5.2.2 Debug Enable Register (DER)

The DER, shown in Figure 44-23, lets the user selectively enable events that can cause the processor to enter debug mode. Its reset value is 0x0200_2000.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—	RSTE	CHSTPE	MCIE	—		EXTIE	ALIE	PRIE	FPUVIE	DECIE	—		SYSIE	TRE	—
Reset	0	0	0	0	00		1	0	0	0	0	0_0		0	0	0
R/W	R/W															
Bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—	SEIE	ITLBMSE	DTLBMSE	ITLBERE	DTLBERE	—							IBRKE	EBRKE	DPIE
Reset	0	0	1	0	0	0	00_0000_0							0	0	0
R/W	R/W															
SPR	149															

Figure 44-23. Debug Enable Register (DER)

DER protection is described in Table 44-15.

Table 44-25. DER Field Descriptions

Bits	Name	Description
0	—	Reserved
1	RSTE	Reset interrupt enable bit 0 Debug mode entry is disabled (reset value) 1 Debug mode entry is enabled
2	CHSTPE	Checkstop enable bit 0 Debug mode entry is disabled 1 Debug mode entry is enabled (reset value)
3	MCIE	Machine check interrupt enable bit 0 Debug mode entry is disabled (reset value) 1 Debug mode entry is enabled
4–5	—	Reserved
6	EXTIE	External interrupt enable bit
7	ALIE	Alignment interrupt enable bit
8	PRIE	Program interrupt enable bit
9	FPUVIE	Floating-point unavailable interrupt enable bit
10	DECIE	Decrementer interrupt enable bit
11–12	—	Reserved
13	SYSIE	System call interrupt enable bit
14	TRE	Trace interrupt enable bit 0 Debug mode entry is disabled 1 Debug mode entry is enabled (reset value)
15–16	—	Reserved
17	SEIE	Software emulation interrupt enable bit 0 Debug mode entry is disabled (reset value) 1 Debug mode entry is enabled
18	ITLBMSE	Implementation-specific ITLB miss enable bit
19	DTLBMSE	Implementation-specific DTLB miss enable bit
20	ITLBERE	Implementation-specific ITLB error enable bit
21	DTLBERE	Implementation-specific DTLB error enable bit
22–27	—	Reserved
28	LBRKE	Load/store breakpoint interrupt enable bit 0 Debug mode entry is disabled 1 Debug mode entry is enabled (reset value)
29	IBRKE	Instruction breakpoint interrupt enable bit
30	EBRKE	External breakpoint interrupt enable bit
31	DPIE	Development port nonmaskable request enable bit

44.5.2.3 Development Port Data Register (DPDR)

The 32-bit development port data register (DPDR), SPR 630, resides in the development port logic. It is used for data interchange between the core and the development system. The DPDR is accessed by using **mtspr** and **mfspr** and implemented using a special bus cycle on the internal bus. See Section 44.3.2.2.1, “Development Port Shift Register.”

Chapter 45
IEEE 1149.1 Test Access Port

The MPC855T 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 to development of this standard under the sponsorship of the Test Technology Committee of IEEE and the Joint Test Action Group (JTAG). The MPC855T implementation supports circuit-board test strategies based on this standard.

The TAP consists of five dedicated signals, a 16-state TAP controller, and two test data registers. A boundary scan register links all device signals into a single shift register. The test logic, implemented using static logic design, operates independently of the device system logic. The MPC855T TAP implementation provides the capability to:

- Perform boundary scan operations to check circuit-board electrical continuity.
- Bypass the MPC855T for a given circuit-board test by effectively reducing the boundary scan register to a single cell.
- Sample the MPC855T system signals during operation and transparently shift out the result in the boundary scan register.
- Disable the output drive to signals during circuit-board testing.

45.1 Overview

The MPC855T TAP implementation includes a TAP controller, a 4-bit instruction register, and two test registers (a 1-bit bypass register and a 475-bit boundary scan register). The TAP interface 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-statable 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 initialization of the TAP controller and other logic required by the standard.

The MPC855T TAP logic is shown in Figure 45-1 below.

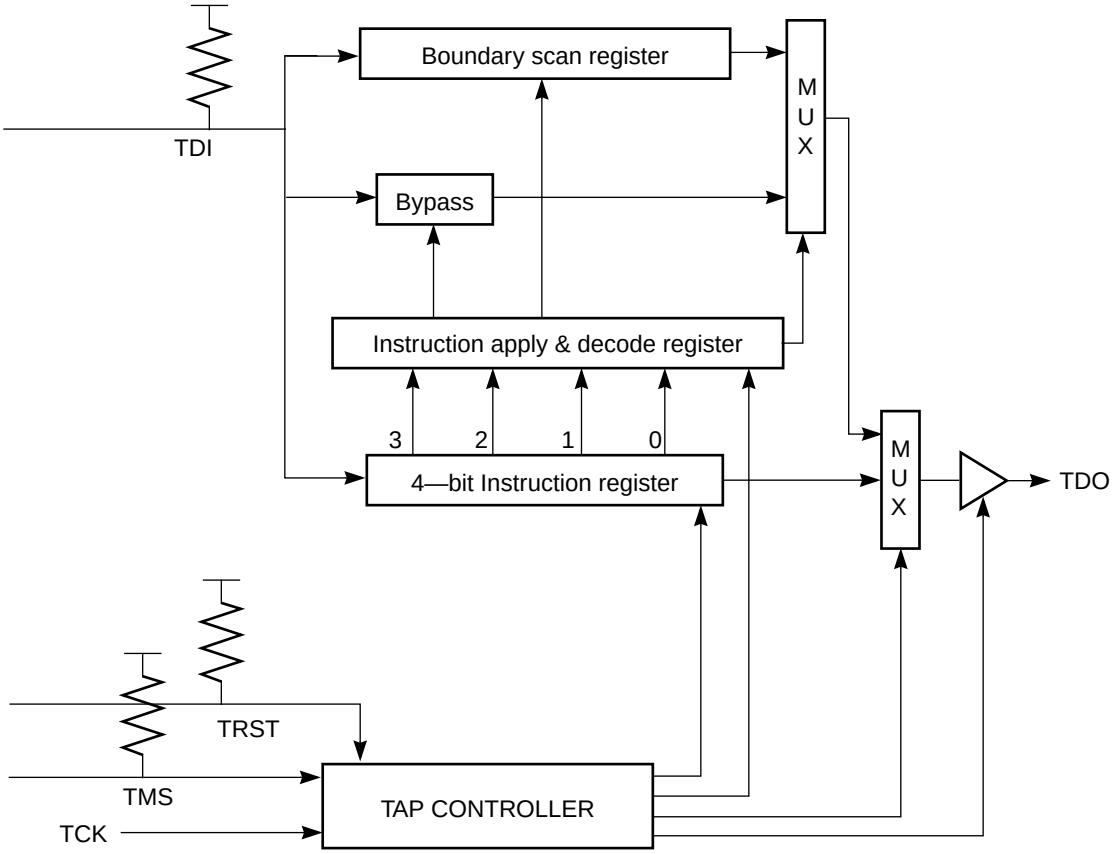


Figure 45-1. Test Logic Block Diagram

45.2 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 represents the value of the TMS signal sampled on the rising edge of TCK. Figure 45-2 shows the MPC855T TAP controller state machine.

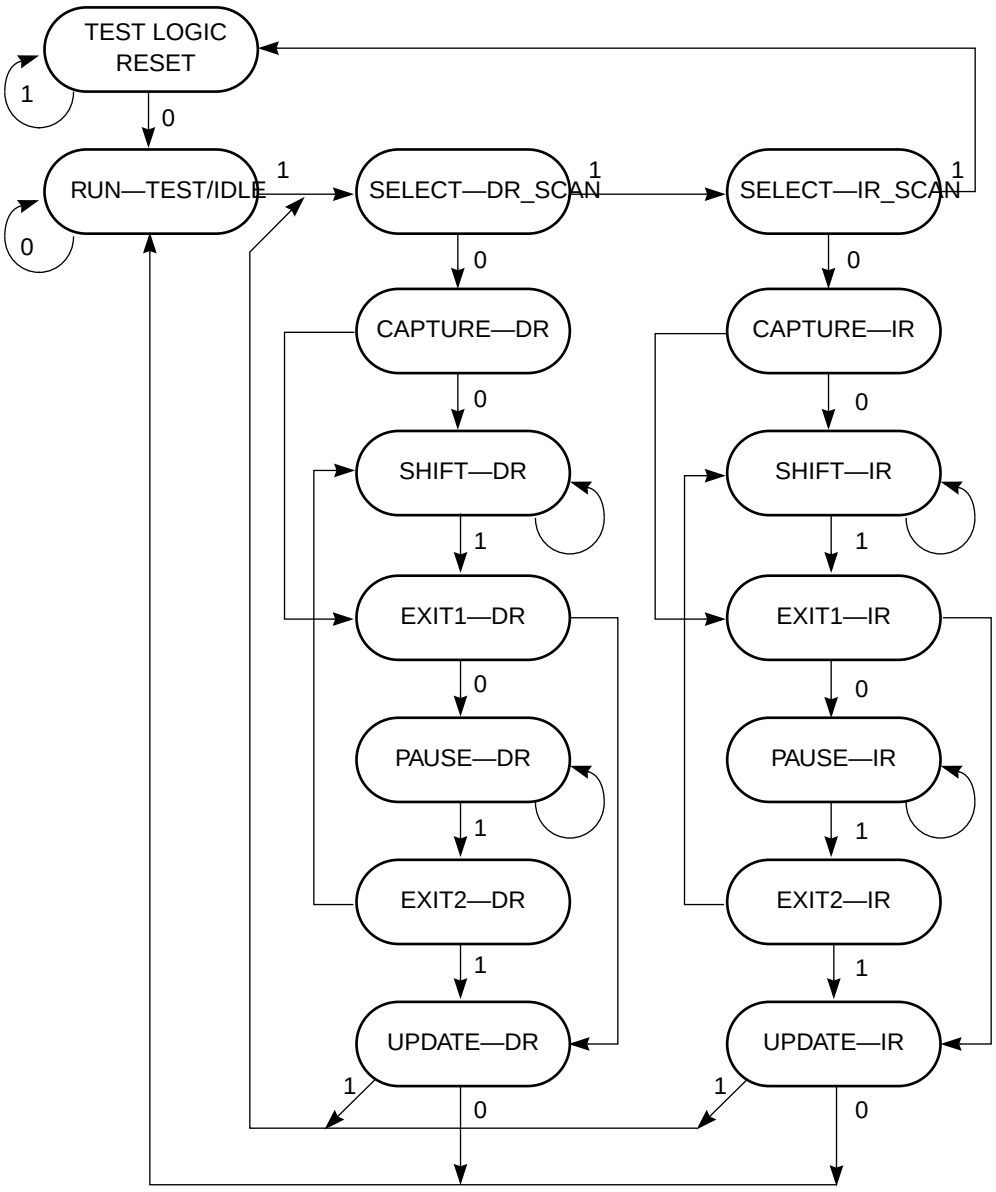


Figure 45-2. TAP Controller State Machine

45.3 Boundary Scan Register

The MPC855T scan chain consists of a 475-bit boundary scan register that contains bits for all signals, with the exception of the XTAL, EXTAL, and XFC pins, which are analog signals. An IEEE-1149.1 compliant boundary scan register has been included on the MPC855T. This 475-bit boundary scan register can be connected between TDI and TDO when EXTEST or SAMPLE/PRELOAD instructions are selected. The boundary scan register is used for capturing data on the input signals, forcing fixed values on the output signals, and selecting the direction and drive characteristics (a logic value or high impedance) of the bidirectional and three-state signals.

Figure 45-3 shows the logic configuration for an output signal boundary scan cell.

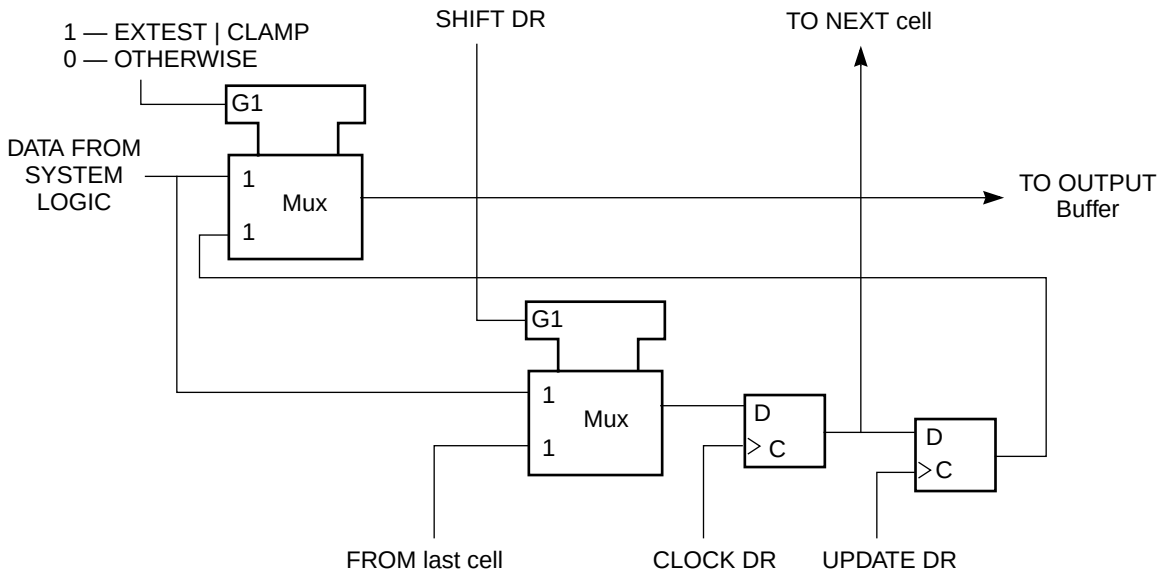


Figure 45-3. Output Signal Boundary Scan Cell (Output Cell)

Figure 45-4 shows the logic configuration for an observe-only input signal boundary scan cell.

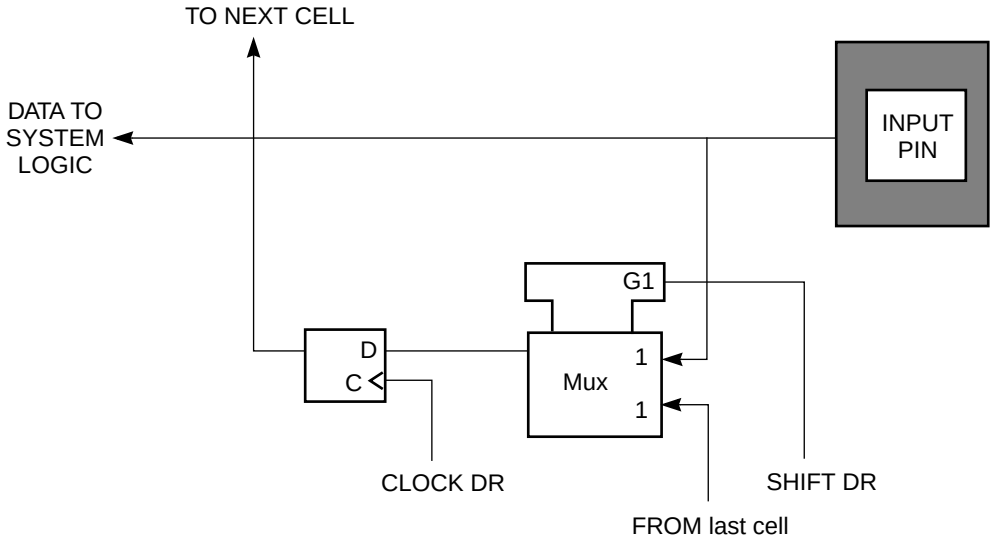


Figure 45-4. Observe-Only Input Signal Boundary Scan Cell (Input Cell)

Figure 45-5 shows the logic configuration for an output control boundary scan cell.

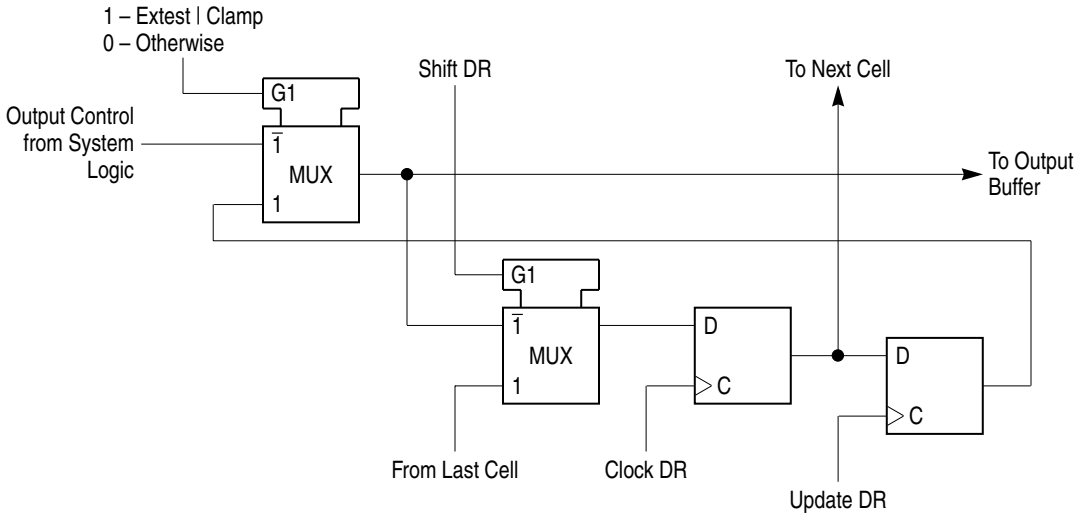


Figure 45-5. Input/Output Control Boundary Scan Cell (I/O Control Cell)

Figure 45-6 shows the logic configuration of bidirectional signal boundary scan cells.

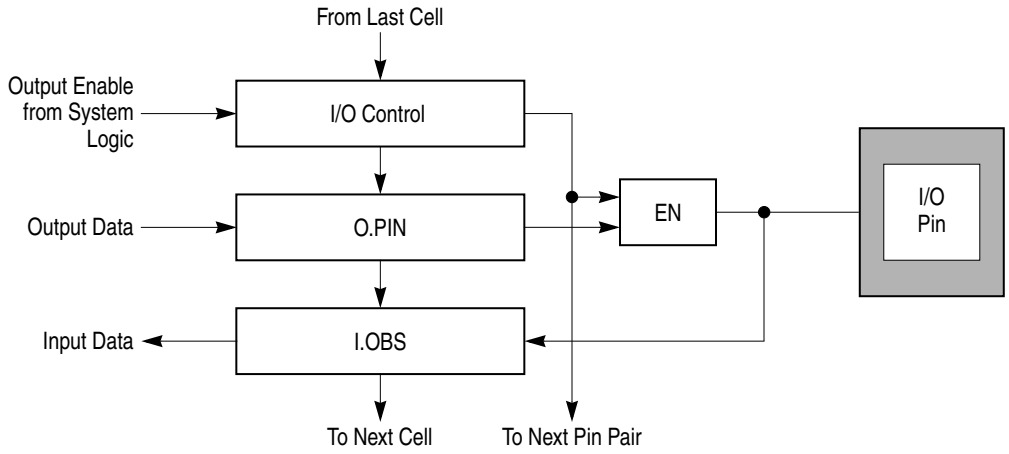


Figure 45-6. Bidirectional (I/O) Signal Boundary Scan Cell

The value of the control bit controls the output function of the bidirectional signal. One or more bidirectional data cells can be serially connected to a control cell. Bidirectional signals include two scan cells for data (input and output buffers) and an I/O control block.

It is important to know the boundary scan bit order and the signals that are associated with them. The bit order of the boundary scan chain (described in the MPC855T BSDL file) starts with the TDO output and ends with the TDI input. The shift register cell nearest TDO (first to be shifted in) is defined as bit 1 and the last bit to be shifted in is bit 475.

45.4 Instruction Register

The MPC855T TAP implementation includes the public instructions EXTEST, SAMPLE/PRELOAD, BYPASS and CLAMP. An optional public instruction (HI-Z)

provides the capability for disabling all device output drivers. The MPC855T TAP implements a 4-bit instruction register (no parity). The 4-bit TAP instructions are executed during the update-IR controller state. The four instruction bits select the five unique instructions listed in Table 45-1.

Table 45-1. Instruction Register 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
0	1	0	1	CLAMP and BYPASS

Note: B0 (lsb) is shifted first.

The instruction register is reset to all ones in the test-logic-reset controller state. During the capture-IR controller state, the inputs to the instruction shift register are loaded with the CLAMP command code.

45.4.1 EXTEST

The external test (EXTEST) instruction enables the 475-bit boundary scan register. EXTEST also asserts an internal soft reset for the MPC855T system logic to force a known beginning internal state while performing external boundary scan operations. Through the TAP, the user is capable of scanning user-defined values into the output buffers, capturing values presented to input pins, and controlling the output drive of three-statable output or bidirectional pins. For more details on the function and use of EXTEST, refer to the IEEE 1149.1 standard.

45.4.2 SAMPLE/PRELOAD

The SAMPLE/PRELOAD instruction initializes the boundary scan register output cells before the boundary scan register is enabled by the EXTEST command. This initialization ensures that known data will appear on the outputs when entering the EXTEST instruction. If the SAMPLE/PRELOAD command was not issued prior to the EXTEST command the output signals will go to a random state when the boundary scan register is enabled and takes control of the output buffer. The SAMPLE/PRELOAD command ensures that the boundary scan register samples the current state of the output signal before it takes control of the associated output buffer. The SAMPLE/PRELOAD instruction also provides an opportunity to obtain a snapshot of system data and control signals.

Note that there is no internal synchronization between the TCK and CLKOUT; the user must provide some form of external synchronization between the JTAG operation at TCK frequency and the system operation CLKOUT frequency to achieve meaningful results.

45.4.3 BYPASS

The BYPASS instruction creates a shift register path from TDI through the bypass register to TDO, circumventing the 475-bit boundary scan register. This instruction is used to enhance test efficiency when a component other than the MPC855T becomes the device under test. The BYPASS instruction selects the single-bit bypass register as shown in Figure 45-7.

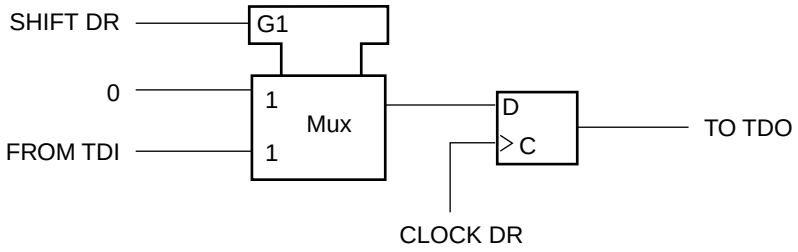


Figure 45-7. Bypass Register

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 TCK in the capture-DR controller state. Therefore, the first bit to be shifted out after selecting the bypass register is always a logic zero.

45.4.4 CLAMP

The CLAMP instruction selects the single-bit bypass register as shown in Figure 45-7 above, and the state of all signals driven from the system output pins is defined by the data currently contained in the boundary scan register.

45.4.5 HI–Z

The HI-Z instruction is provided as a manufacturer's optional public instruction to avoid back driving the output pins during circuit-board testing. When the HI-Z instruction is invoked all output drivers, including the two-state drivers, are placed in a high impedance state. The HI-Z instruction also selects the bypass register.

45.5 TAP Usage Considerations

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. The user must avoid situations in which the MPC855T output drivers are enabled into actively driven networks.

45.6 Recommended TAP Configuration

To ensure that the scan chain test logic is kept transparent to the system logic during normal operation, the TAP should be forced into the test-logic-reset controller state by keeping $\overline{\text{TRST}}$ or TMS continuously asserted.

The TAP signals must be configured as follows to reset the scan chain logic:

- If both the TAP and low power mode are never used, connect $\overline{\text{TRST}}$ to ground.
- If the TAP or low power mode is used, connect $\overline{\text{TRST}}$ to $\overline{\text{PORESET}}$.
- If power down mode (the lowest power mode, where V_{DDH} is disabled) is used, connect $\overline{\text{TRST}}$ to $\overline{\text{PORESET}}$ through a diode (anode to $\overline{\text{TRST}}$, cathode to $\overline{\text{PORESET}}$).

The TMS, TDI, and $\overline{\text{TRST}}$ signals include on-chip pull-up resistors. TCK, however, does not have an on-chip pull-up or pull-down resistor; it should be pulled down through a resistor.

To use the TAP to perform test operations, select the TAP functions in the hard reset configuration word for the signals TCK/DSCK, TDI/DSDI, TDO/DSDO; see Section 11.1.3.1, “PLL Loss of Lock.”

45.7 Motorola MPC855T BSDL Description

The most current revision of the BSDL file for the MPC855T PowerQUICC is available at the Motorola web site (www.motorola.com).

Appendix A
Byte Ordering

This microprocessor supports three byte-ordering conventions—big-endian (BE), true little-endian (TLE), and modified little-endian (MOD-LE). This chapter describes each of the three endian modes. Chapter 3, “Operand Conventions,” in *Programming Environments Manual for 32-bit Implementations of the PowerPC Architecture*, provides a general overview of byte ordering.

A.1 Byte Ordering Overview

For big-endian byte ordering, the most-significant byte (MSB) is stored at the lowest address while the least-significant byte (LSB) is stored at the highest address. This is called big-endian because the big end of the scalar comes first in memory.

For true little-endian byte ordering, the LSB is stored at the lowest address while the MSB is stored at the highest address. This is called true little-endian because the little end of the scalar comes first in memory.

For modified little-endian byte ordering (also referred to as ‘munged little-endian’), the address of data is modified so that the memory structure appears little-endian to the executing processor, when in fact, the byte ordering is big-endian. The address modification is called ‘munging’. Note that the term ‘munging’ is not defined or used in the PowerPC architecture specification.

A.2 Byte-Ordering Mechanisms

There are several byte-ordering mechanisms that are controlled by programmable parameters. The MSR[LE] and MSR[ILE] bits control a 3-bit address modifier in the MPC8xx core. The DC_CST[LES] bit controls a 2-bit address modifier in the core and a 2-bit address modifier and byte lane swapper in the SIU. The FCR[BO] field of each peripheral (SCC, SMCs, SPI, I^2C, PIP, or IDMA) controls a 3-bit address modifier in the SDMA. Table A-1 correlates the programmable parameters with the byte-ordering modes of operation.

Table A-1. Byte-Ordering Parameters

Byte-Ordering Mode	Parameter		
	MSR[LE] or MSR[ILE]	DC_CST[LES]	FCR[BO]
BE	0	0	1x
TLE	0	1	1x
MOD-LE	1	0	01

Note: The powers up in BE mode.

A.3 BE Mode

As shown in Table A-1, the MPC855Tpowers up in BE mode. In BE mode, the caches, internal registers, the U-bus, and the external bus, all use big-endian byte ordering. In BE mode, no address modification nor data-byte-lane swapping is performed by any of the byte-ordering mechanisms of theMPC855T.

The PowerPC architecture defines two bits in the MSR for specifying byte ordering—LE (little-endian mode) and ILE (exception little-endian mode). In this microprocessor, these bits only control the addresses generated by the MPC8xx core. The LE bit specifies the endian mode for normal core operation and ILE specifies the mode to be used when an exception handler is invoked. That is, when an exception occurs, the ILE bit (as set for the interrupted process) is copied into MSR[LE] to select the endian mode for the context established by the exception. For both bits, a value of 0 specifies BE mode (or TLE mode, depending on DC_CST[LES]), and a value of 1 specifies MOD-LE mode.

A.4 TLE Mode

When operating in TLE mode, the external bus uses little-endian byte ordering, so any external agents should use little-endian byte ordering to access memory. Note however, that internal to the microprocessor, the caches and internal registers use big-endian byte ordering. The byte-ordering mechanisms for TLE mode are shown in Figure A-1.

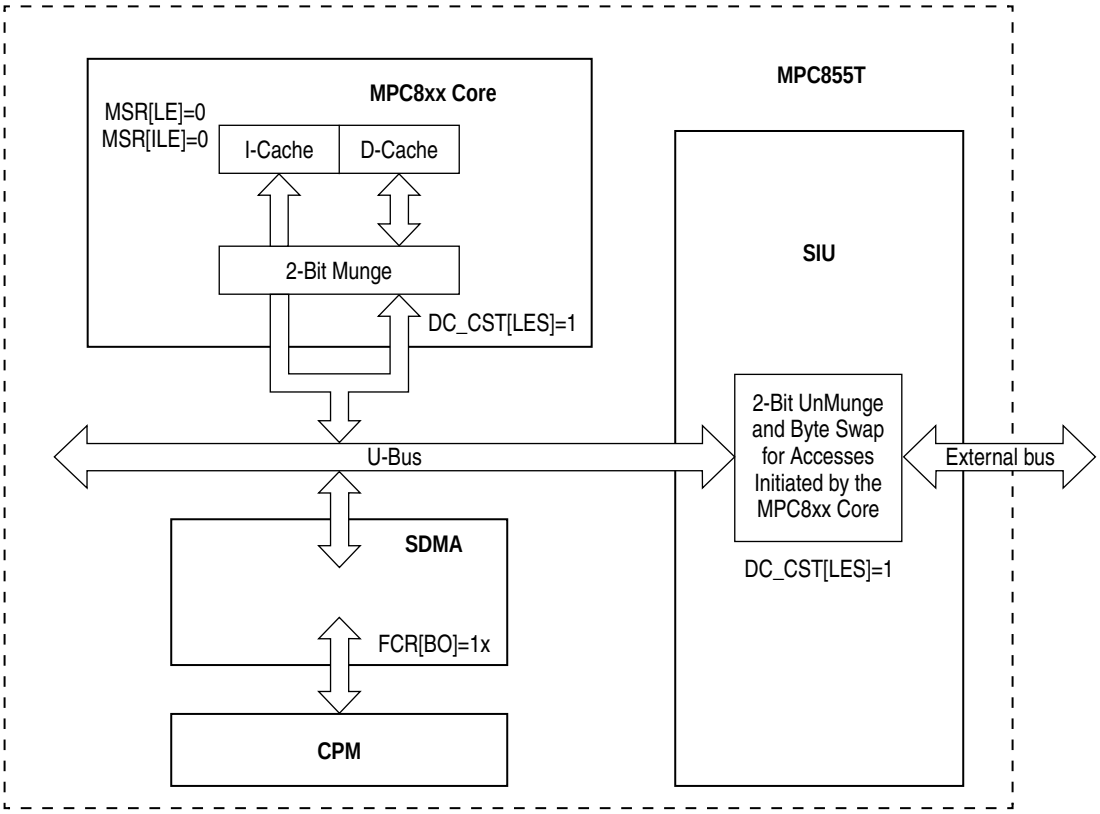


Figure A-1. TLE Mode Mechanisms

For TLE mode, MSR[LE] and MSR[ILE] should be cleared as in BE mode. (This disables the 3-bit address munging used in MOD-LE mode. See Section A.5, “MOD-LE Mode,” for more information.)

For TLE mode, DC_CST[LES] should be set. When DC_CST[LES] is set, the physical address is modified before the data cache or load/store unit accesses the internal U-bus. The two low-order address bits of the effective address are exclusive-ORed (XOR) with a two-bit value that depends on the length of the operand (1, 2, or 4 bytes), as shown in Table A-2. This process is called 2-bit munging.

Table A-2. TLE 2-bit Munging

Data Width (Bytes)	Address Modification
4	No change
2	XOR with 0b10
1	XOR with 0b11

Since all instructions are 4 byte words, no address modifications by the instruction cache are necessary.

The munged physical address is passed to the internal U-bus, and the specified width of data is transferred. Only the address is modified, not the byte order. Munging makes it appear to the core that individual aligned scalars on the U-bus are in little-endian order, when in fact, they are actually in big-endian order. This allows the core to access data in the inherently big-endian internal registers with apparent little-endian byte-ordering. However, when DC_CST[LES] is set, for any access originating from the MPC8xx core, the SIU unmunges the address and swaps the bytes of data within each word at the external bus/U-bus boundary. The byte swapping is shown in Figure A-2.

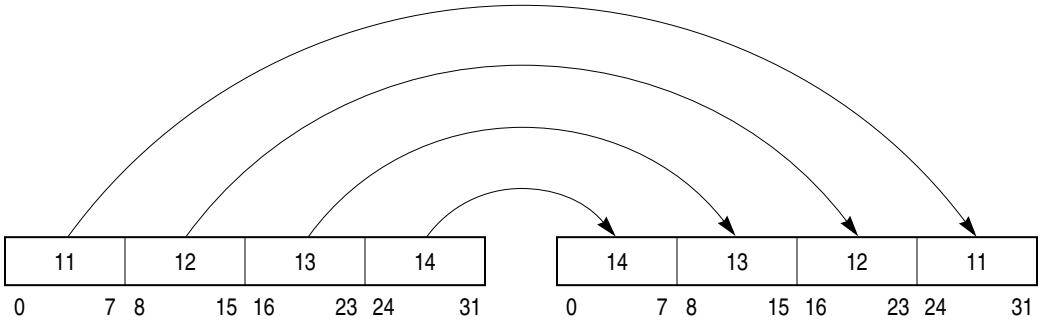


Figure A-2. Byte Swapping

The unmunging and byte swapping places all external accesses by the MPC8xx core into true little-endian byte order. Note that the bit ordering remains unchanged—that is, bit 0 is always the msb, and bit 31 is always the lsb.

The communication peripherals (SCCs, SMCs, SPI, I^2C, PIP, or IDMA) transfer data as bytes (bytes are received one at a time and transmitted one at a time). Byte transfers have no inherent endianess—they are neither big- nor little-endian. For TLE-mode, the FCR[BO] parameter of each peripheral should be programmed to 0b1*x* (that is, either 0b10 or 0b11). Note that the SIU does nothing (no unmunging, no byte-swapping) to accesses originating from the SDMA controller.

A.4.1 TLE Mode System Examples

The following tables describe how to handle the little-endian program or data in the little-endian system that is built around the MPC855T for various port sizes.

Table A-3. Little-Endian Program/Data Path Between the Register and 32-Bit Memory

Fetch/ Load Store Type	Little-Endian Addr	U-bus and Cache Addr	External Bus Addr	Data in the Register				U-bus and Cache Format				External Bus Format				Little-Endian Program/Data			
				MSB			LSB	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 A-4. Little-Endian Program/Data Path Between the Register and 16-Bit Memory

Fetch/ Load Store Type	Little-Endian Addr	U-bus and Cache Addr	External Bus Addr	Data in the Register				U-bus and Cache format				External Bus Format				Little-Endian Program/Data			
				MSB			LSB	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 A-5. Little-Endian Program/Data Path between the Register and 8-Bit Memory

Fetch/ Load Store Type	Little-Endian Addr	U-bus and Cache Addr	External Bus Addr	Data in the Register				U-bus and Cache Format				External Bus Format				Little-Endian Program/Data			
				MSB			LSB	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’

A.5 MOD-LE Mode

For modified little-endian (MOD-LE) mode, the caches, the U-bus, and the external bus use big-endian byte ordering with munged addresses. The byte-ordering mechanisms for MOD-LE mode are shown in Figure A-3.

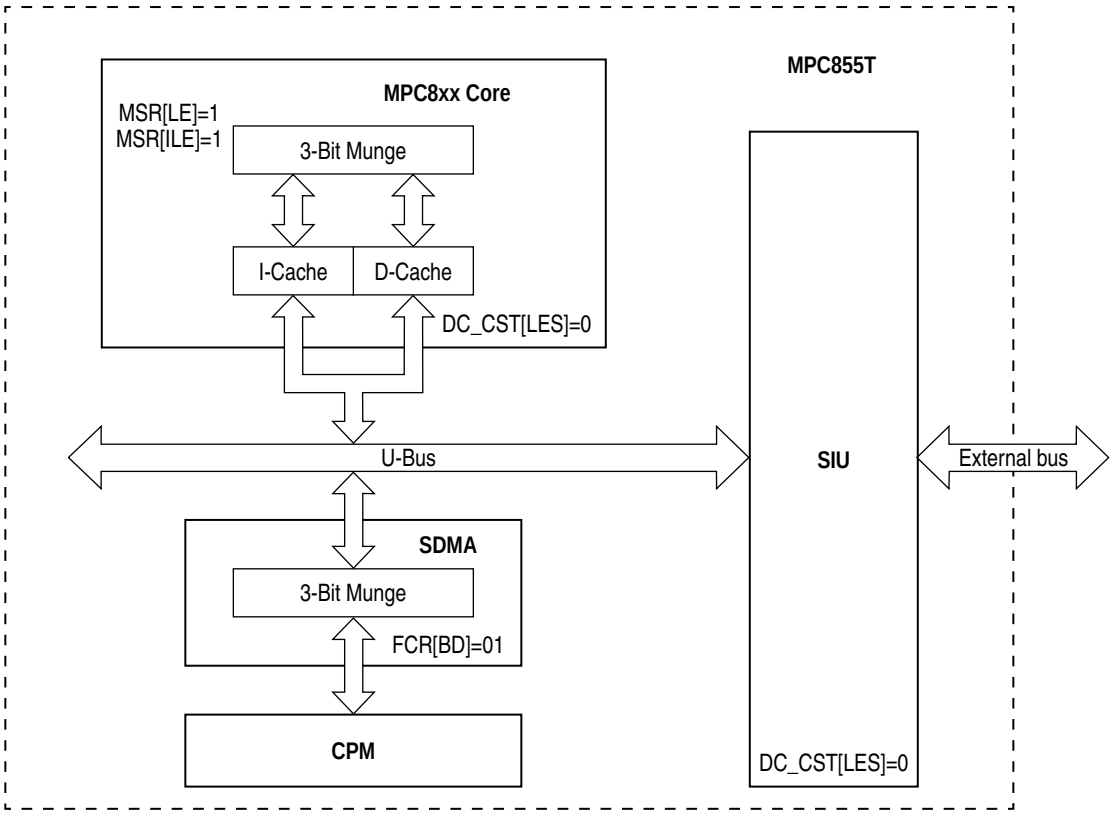


Figure A-3. MOD-LE Mode Mechanisms

For MOD-LE mode, MSR[LE] and MSR[ILE] should be set. When the MSR[LE] is set, the effective address is modified (munged) by the core before being used to access the caches (or memory for caching-inhibited accesses). The three low-order address bits of the effective address are exclusive-ORed (XOR) with a three-bit value that depends on the length of the operand (1, 2, or 4 bytes), as shown in Table A-6 This process is called '3-bit munging.'

Table A-6. MOD-LE 3-bit Munging

Data Width (Bytes)	EA Modification
4	XOR with 0b100
2	XOR with 0b110
1	XOR with 0b111

Note that theMPC855T does not support 8-byte scalars

The munged physical address is passed to the cache or to external memory, and the specified width of data is transferred (in big-endian order—that is, MSB at the lowest address, LSB at the highest address). In MOD-LE mode, only the address is modified, not the byte order. Munging makes it appear to the core that individual aligned scalars are stored in little-endian order, when in fact, they are stored in big-endian order, but at

different byte addresses within double words. Note that the instruction and data caches operate less efficiently when address munging is performed on cache accesses. Some performance degradation should be expected when the MPC855T is operating in MOD-LE mode

MSR[ILE] is used to set the endian mode of the core during exception handling. When an exception occurs, MSR[ILE] is copied into MSR[LE] to select the endian mode for the context established by the exception.

For MOD-LE-mode, the FCR[BO] parameter of each peripheral (SCCs, SMCs, SPI, I^2C, PIP, or IDMA) should be set to 0b01. The SDMA controller examines the BO parameter and, if set to 0b01, performs a 3-bit munge (XOR with 0b111) on every byte address of transmitted or received data as in Table A-6.

A.5.1 I/O Addressing in MOD-LE Mode

For a system running in BE or TLE mode, both the MPC855T and the memory subsystem recognize the same byte as byte 0. However, this is not true for a system running in MOD-LE mode because of the munged address bits when theMPC855T accesses external memory.

For I/O transfers in MOD-LE mode to transfer bytes properly, they must be performed as if the bytes transferred were accessed one at a time, using the little-endian address modification appropriate for the single-byte transfers (that is, the lowest order address bits must be XORed with 0b111). This does not mean that I/O operations in MOD-LE systems must be performed using only one-byte-wide transfers. Data transfers can be as wide as desired, but the order of the bytes within double words must be as if they were fetched or stored one at a time. That is, for a true little-endian I/O device, the system must provide a mechanism to munge and unmunge the addresses and reverse the bytes within a doubleword (MSB to LSB).

A load or store that maps to a control register on an external device may require the bytes of the register data to be reversed. If this reversal is required, the load and store with byte-reverse instructions (**lhbrx**, **lwbrx**, **sthbrx**, and **stwbrx**) may be used.

A.6 Setting the Endian Mode Of Operation

As shown in Table A-1, the MPC855T powers up in BE mode. The endian mode should be set early in the initialization routine and remain unchanged for the duration of system operation. 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 switch the system from BE to MOD-LE mode, the MSR[LE] and MSR[ILE] bits should be set using an **mtmsr** instruction that resides on an odd word boundary (A[29] = 1). The instruction that is executed next will be fetched from this address plus 8. If the **mtmsr**

instruction resides on an even word boundary (A[29] = 0), then the instruction will be executed twice due to the address munging of MOD-LE mode.

To switch the system from MOD-LE to BE mode, the MSR[LE] and MSR[ILE] bits should be cleared using an **mtmsr** instruction that resides on an even word boundary (A[29] = 0). The instruction that is executed next will be fetched from this address plus 12.

To switch the system to TLE mode, DC_CST[LES] should be set using an **mtspr** instruction that resides on an even word boundary (A[29] = 0). Further instructions should reside in the little-endian format of the external system memory or in the big-endian format of the internal memory (if it exists).

The buffer descriptors for the peripherals contain the FCR[BO] parameters for the SDMA controller. The BO parameter should be set to the required endian format prior to activating the associated peripheral.

Appendix B
Serial Communications Performance

Due to the architecture of the MPC855T, the overall performance of the serial channels cannot be stated in absolute terms. The serial channels of the MPC855T can be programmed in many different modes, which require different degrees of processing. There may be several individual bottlenecks in the system, with their own specific considerations. These are described in the following sections.

Note that since the MPC855T has only one SCC, CPM overloading may not be a problem. ATM, receiver, and transmitter data is therefore given as it applies to the MPC860. This appendix is provided as an example to check overall system design.

B.1 Serial Clocking (Peak Rate Limitation)

The maximum rate at which a serial channel can be clocked is governed by the synchronization hardware of the serial channels. The rate at which the serial channel can be clocked depends on the physical interface of the channel. Examples include:

- maximum clocking rate for an SCC connected to a dedicated set of pins (NMSI mode)=SYNCCLK/2
- maximum clocking rate for the TDM channel=SYNCCLK/2.5

For limitations of other channels, refer to the appropriate chapter of the manual.

SYNCCLK is a programmable clock rate which is derived from the system frequency; see Chapter 14, “Clocks and Power Control.” At its maximum rate, it is equal to the system frequency.

The maximum serial clock rate is a limitation on the peak data rate. This is the maximum rate at which the receiver or transmitter hardware can transfer data between its internal FIFO and the serial line. However, this rate is higher than the rate at which data in these internal FIFOs can be processed by the CPM and transferred to system memory. Therefore, this peak rate can only be maintained for short bursts which do not exceed the internal FIFO depth. The serial clocks must also be turned off between these bursts. The FIFOs of SCC1 is 32 bytes. The SMCs and SPI are double-buffered, and thus have an effective FIFO depth of two characters.

To summarize, the architecture of the MPC855T allows the serial channels to handle high-speed bursts of data for short periods of time subject to their internal FIFO sizes. If transfers are sufficiently short, and if serial clocks are turned off between the transfers, then these individual transfers can be performed at up to the peak rate. Over time, however, the average amount of data transferred must not exceed the average CPM processing rate.

If any of the conditions outlined above are not satisfied, then the rate at which the serial channels are clocked must not exceed the rate at which the CPM can process data from them. In other words, the average rate limitation must also be treated as the peak rate limitation.

The I^2C channel is the only exception to these rules. Its maximum serial transfer rate is limited by its hardware, not by the rate at which the CPM services it. At its maximum transfer rate, it will only consume 25% of the CPM bandwidth.

B.2 Bus Utilization

Given the width and clock speed of the system bus of the MPC855T, bus utilization is not a critical system limitation, considering the data rates supported by the MPC855T. Specifically, the peak system bus transfer rate of a 50MHz MPC855T (using single-beat transfers to zero wait-state memory) is 800Mbps, whereas the maximum aggregate serial data rate supported at that frequency is usually less than 50Mbps.

However, whereas bus utilization is not a major consideration, bus latency can be. Extreme periods of bus latency could potentially cause a FIFO to overrun or underrun. Where this is a more critical issue, some specific recommendations are made. For example, recommendations for system bus latency are made for an MPC860MH operating in QMC mode; see the *QMC Supplement to MC68360 and MPC860 User's Manuals*.

B.3 CPM Bandwidth (Average Rate Limitation)

The communications processor module (CPM) is a single shared resource used by all of the serial channels. It handles low-level protocol processing tasks and manages DMA for all of them. In the architecture of the MPC855T, the CPM is the central communications processing engine, to which the individual serial controllers (SCC, SMC, SPI, I^2C, and PIP) make requests for service. Therefore, the serial channels must not request more service than the CPM can provide; else, FIFO underrun or overrun errors will result.

The amount of processing required by a particular serial channel depends on the mode in which the channel is configured, and the maximum rate at which the channel requests service. [While this rate is usually equivalent to the serial clocking rate, under certain conditions the serial clocking rate could be higher; this is because the FIFOs of the serial channels can provide a ‘local averaging’ effect on the data rate, and thus can handle short bursts. See Section B.1, “Serial Clocking (Peak Rate Limitation).”]

B.3.1 Performance of Serial Channels

The table provided in this section lists the data rates supported by the CPM for particular channels in different modes. These figures assume that the serial channel in question is the only channel in operation. Individual channels operating at the data rate quoted would consume 100% of the CPM bandwidth.

The performance available from different serial channels in different protocols varies greatly. This is due to the amount of overall processing required by the protocol and by the split between hardware-assist provided in the serial channel and processing performed in the CPM by microcode. For example:

- An SCC in UART mode provides more processing in the SCC hardware, whereas an SMC in UART mode is more reliant on the CPM. Therefore, the performance of an SCC in UART mode is greater.

An SCC in HDLC mode performs most of the processing (e.g. bit manipulation, deframing) in hardware, whereas HDLC processing for QMC channels falls on the communications processor module (CPM). Thus, an SCC in HDLC mode can process more data than an SCC in QMC mode, even if all QMC time slots are concatenated into one logical channel.Maximum data rates are given for most channels as full duplex. Channels operating in half duplex will require only half the CPM service, and thus the maximum data rates supported for these channels doubles.

Managing DMA for the serial channels is a significant portion of the CPM processing. Therefore, because channels with larger frame sizes require the CPM to access the buffer descriptors less often, these channels experience higher performance. An example of this shown in the table below is an SCC in HDLC mode; a channel with a minimum frame size of 64 bytes has better performance than one with a minimum frame size of 5 bytes.

The performance figures listed in Table B-1. are for a 25 MHz system clock only. In general, performance scales linearly with frequency; an MPC855T with a 50 MHz system clock would support twice the quoted data rate. Thus, a combination of serial channels and protocols which are beyond the MPC8’s performance scope at 25 MHz may be possible at 50 MHz.

Performance figures quoted in Table B-1 assume worst-case conditions. Worst-case conditions are a steady stream of minimum-size frames.Furthermore, for the SCC in QMC mode, it assumes that all virtual channels simultaneously reach end-of-frame, and thus all must close and open buffers simultaneously.

Table B-1. MPC855T Serial Performance at 25 MHz

Protocol	Speed [see note 2]
SCC in transparent	8 Mbps FD
SCC in HDLC (5 byte minimum frame size)	8 Mbps FD
SCC in HDLC (64 byte minimum frame size)	11 Mbps FD

Table B-1. MPC855T Serial Performance at 25 MHz

Protocol	Speed [see note 2]
SCC in UART	2.4 Mbps FD
SCC in Ethernet	22 Mbps HD
SCC in Ethernet	11 Mbps FD
SCC in QMC mode	2.1 Mbps FD
SCC in BISYNC	1.5 Mbps FD
SCC in asynchronous HDLC/IrDA	3 Mbps FD
SMC in transparent	1.5 Mbps FD
SMC in UART	220 Kbps FD
I^2C	520 Kbps [see note 1]
SPI (16 bit)	3.125 Mbps
SPI (8 bit)	500 Kbps
PIP (8 bit width)	625 Kbyte/s
PIP (16 bit width)	1250 Kbyte/s
SCC in Profibus [optional RAM microcode]	2.4 Kbps FD
SCC in SS#7 [optional RAM microcode]	6 Mbps FD
SCC in SS#7 [optional RAM microcode] [without scrambling]	8 Mbps FD
SCC in SS#7 [optional RAM microcode] [with scrambling]	5.5 Mbps FD

Note:

1. I^2C is a special case. Its performance is limited by its hardware, not by the CPM. An I^2C port operating at 520 Kbps would consume only 25% of the CPM bandwidth of an MPC855T operating at 25 MHz.
2. Performance scales linearly with system frequency.
3. FD indicates full-duplex; HD indicates half-duplex.
4. Ethernet full and half duplex modes are quoted separately merely to highlight the feature.
5. SPI is inherently full-duplex, and it is therefore not necessary to mark it as so.

B.3.2 IDMA Considerations

Although the IDMA channels are implemented in microcode by the CPM, they need not necessarily be calculated into the CPM bandwidth. If IDMA is not a time-critical task, then its priority can be programmed to be the lowest of the CPM tasks. If this is done, IDMA is treated as a background task and serviced only when other channels do not require service.

If IDMA is configured to be a higher-priority task, then its transfers must be considered when calculating demands on the CPM bandwidth. Table B-2 provides performance

information for the IDMA channels in their different modes. Its use is similar to the table provided for the serial channels (Table B-1.).

Table B-2. IDMA Performance at 25 MHz

Protocol	Speed
IDMA memory to memory	5.7 MByte/s
IDMA memory to memory with burst aligned source/dest address	10.4 MByte/s
IDMA dual address, peripheral to memory	2.2 MByte/s
IDMA dual address, memory to peripheral	1.6 MByte/s
IDMA single address, peripheral to memory	5 MByte/s
IDMA single address, memory to peripheral	5 MByte/s

Note:

1. Performance scales linearly with device operating frequency.
2. IDMA transfer rates are independent of bus cycle length.

B.3.3 Performance Calculations

Special configurations verified by experiment.

The performance figures quoted in Table B-1. can be used to estimate the overall CPM bandwidth required in a particular system configuration. To calculate the total system load, add the CPM utilization from every channel together. Assuming approximately linear performance versus frequency, the general problem reduces to taking simple ratios:

$$\text{CPM Utilization} = \left(\frac{\text{serial rate}_1}{\text{max serial rate}_1}\right) + \left(\frac{\text{serial rate}_2}{\text{max serial rate}_2}\right) \cdots$$

For example, since a 25-MHz MPC855T running Ethernet (theoretically) at 22 Mbps consumes approximately 100% of the CPM bandwidth, what bandwidth does a (practical) 10-Mbps channel require?

$$\text{CPM Utilization} = \frac{\text{serial rate}}{\text{max serial rate}} = \frac{10}{22} = 0.45$$

The above equation shows the 10-Mbps channel requiring 45% of the CPM bandwidth of a 25-MHz MPC855T.

A spreadsheet tool for performing serial performance calculations has been developed and is available on the world-wide web site at http://www.motorola.com. It is entitled “CPM Performance Spreadsheet.”

Please note that CPM load estimation is a linear approximation to a somewhat nonlinear phenomenon, and cannot be relied upon to be exact. When performance estimations

approach the maximum loading (i.e. greater than approximately 95%), the user should test the system on target hardware to determine the exact load. Conversely, some system configurations that calculate to greater than 100% by these equations have been verified by experiment. These include the following:

-
  - A 25 MHz MPC860MH with one half-duplex Ethernet and 24 QMC channels (i.e. 24 x 64 kbps QMC) [if and only if SCC1 is Ethernet and QMC channels are spread over two SCCs, e.g. SCC1=Ethernet, SCC2=QMC channels 0-11, SCC3=QMC channels 12-23].
  - A 33 MHz MPC860MH with one half-duplex Ethernet and 32 QMC channels (i.e. 32 x 64 kbps QMC) [if and only if SCC1 is Ethernet QMC channels are spread over two SCCs, e.g. SCC1=Ethernet, SCC2=QMC channels 0-15, SCC3=QMC channels 16-31].
  - A 40 MHz MPC860EN with four half-duplex Ethernet channels.

More examples of CPM bandwidth calculations follow:

Example #1:

MPC860 (at 25 MHz) operating 1 × 10 Mbps Ethernet in half duplex, 1 × 2 Mbps HDLC, 1 × 64 Kbps HDLC, 1 × 9.6 Kbps UART and 1 × 38 Kbps SMC UART. The following equation applies:

$$\left(\frac{10}{22}\right)+\left(\frac{2}{8}\right)+\left(\frac{0.064}{2.4}\right)+\left(\frac{0.0096}{2.4}\right)+\left(\frac{0.038}{0.22}\right) = 0.89 \qquad \textbf{(<1)}$$

This yields a percentage CPM utilization of 89% meaning the device can handle these protocols at this frequency. Note the 9.6-Kbps UART link only requires 0.4% of the CPM bandwidth, implying that in any configuration where there is free bandwidth that it will be possible to run a low-rate UART link.

Example #2:

MPC860MH (at 25 MHz) running 24 QMC channels with an additional 2 HDLC channels operating at 128 Kbps each. The following equation applies:

$$\left(\frac{2\times 0.128}{8}\right)+\left(\frac{24\times 0.064}{2.1}\right) = 0.76 \qquad \textbf{(<1)}$$

Example #3:

MPC860MH (at 25 MHz) running 32 QMC channels and one additional 2 Mbps HDLC channel. The following equation applies:

$$\left(\frac{2}{8}\right) + \left(\frac{32 \times 0.064}{2.1}\right) = 1.22$$ **(will not work)**

Since the result above is greater than one, this will not work with a 25-MHz MPC860MH. However, if the system clock is increased to 33 MHz, CPM utilization drops below 1, allowing example #3 to be supported. The following equation applies:

$$1.22 \times \left(\frac{25}{33}\right) = 0.92 \qquad \textbf{(<1)}$$

Example #4:

MPC860 (at 25 MHz) with a block of data transferred by IDMA at 512 Kbytes/s to a 32-bit peripheral, one asynchronous HDLC at 1Mbps, one UART at 9,600 baud, and one transparent channel at 2 Mbps.

$$\frac{0.512}{5} + \frac{1}{3} + \frac{0.0096}{2.4} + \frac{2}{8} = 0.69$$

In the case of IDMA, this process calculates the peak CPM utilization, not the sustained rate. By nature, IDMA transfers occur at random intervals and are not consistent bit rates when compared to the serial channel operation.

Example #5:

MPC860 (at 40 MHz) with three Ethernet channels at 10 Mbps and one UART at 9,600 baud.

$$\left[\frac{3*10}{22} + \frac{0.0096}{2.4}\right] = 1.37 \times \frac{25}{40} = \mathbf{0.85}$$

B.4 ATM Performance

This appendix provides receiver and transmitter performance information for the MPC855T.

The information was gathered under the following conditions:

- System clock = 50MHz
- AAL5 - Buffer (data) size > 200 bytes
- AAL0 - Interrupt per BD
- Average load on external bus
- System memory is DRAM with 5-2-2-2 performance at 50MHz.

NOTE

CPM performance is theoretically linear to the system clock. However, a slow external memory or an overloaded PPC bus can degrade the performance figures.

ATM performance is also influenced by the number of PHYs and the timer 4 rate for the APC. The more PHYs are serviced with the APC or the higher the timer 4 rate is configured, the less the maximum bit rate for ATM can be. Please contact your Motorola representative for more information.

B.5 Receiver

Table B-3 shows the UTOPIA and serial ATM receiver performance of the MPC855T when configured with internal and external connection tables.

Table B-3. Receiver Performance (with 50MHz System Clock)

Mode	Condition	Performance (in Mbps)	
		UTOPIA	Serial (3)
Internal Channels	AAL5, middle frame, look-up table (1)	89	27/19
	AAL0, no CRC10, look-up table (1)	68	24/17.5
	AAL5, middle frame, address comp (2)	84	26/18
	AAL0, no CRC10, address comp (2)	65	23/17
External Channels	AAL5, middle frame, address compression	69	23/18
	AAL0, Address compression	50	21/17
	AAL5, middle frame, CAM	84	27/19
	AAL0, CAM	57	24/17.5
External AAL5 + MPHY	AAL5, middle frame, Address compression	66	NA

1. Cell header located in the middle of the look-up table (16th place), AAL0 RCT[NCRC] is set.
2. Address compression with FLMASK and SLMASK containing 4 lsb zeros.
3. No scrambler / With Scrambler and coset

The following table allows for calculating the impact of several modes on performance.

Table B-4. Additional Features Load

Mode	Numerator	Denominator (CPM Load in Mbps)
MCF set	Total RX bit rate	1678
CRC-10 (in AAL0 channel)	Total bit rate of AAL0 channels with CRC10	457

Table B-4. Additional Features Load (continued)

Mode	Numerator	Denominator (CPM Load in Mbps)
PM on internal channel	Total bit rate of internal channels with PM	694
PM on external channel	Total bit rate of external channels with PM	419
Statistics mode activated, single PHY	Total RX bit rate	2870
Statistics mode activated, multy PHY	Total RX bit rate	1340

Example:

The following load exists:

10 AAL0 internal channel, each receives 5Mbps.

2 AAL5 internal channels, 10Mbps each, and one of the channels has PM on.

What is the overall CPM load?

the load caused by the AAL0 channels = total_AAL0_rate/max_AAL0_rate = (10*5)/68 = 50/68

the load caused by the AAL5 channels = total_ALL5_rate/max_AAL5_rate = (2*10)/89

the load caused by the PM processing = 10/694

overall load = 50/68 + 20/89 + 10/694 = 0.97 < 1

We see that in the PM load, the numerator contains only the bit rate that carries PM (only 1 AAL5 channel performs PM hence only 10 Mbps).

The overall bit rate came out less than one. This means that the CPM is able to handle the calculated load.

If in addition to the previous channels configuration, MCF is also activated, the new CPM load would yield:

the load caused by MCF = total_channel_rate/MCF_max_rate = (10*5+2*10)/1678

And the overall load is 0.97 + 70/1678 = 1.01

This time the overall load is higher than one. This means that we have exceeded the CPM capacity. This can be resolved by various ways: raising the system clock, improving external memory, or cancelling some of the traffic.

B.6 Transmitter

Table B-5 shows the UTOPIA and serial mode transmitter performance of the MPC855T when configured with internal and external connection tables.

Table B-5. Transmitter (Including 1 Priority APC) Performance (with 50MHz System Clock)

Mode	Condition	Mbit	
		UTOPIA(4)	Serial(5)
INTERNAL CHANNELS	AAL5, middle frame + APC	57/69	32/23
	AAL0 +APC	51/60	28/20.5
EXTERNAL CHANNELS	AAL5, middle frame + APC	47/51	27/20
	AAL0 + APC	43/47	24/19
EXTERNAL AAL5 + MPHY	2 MPHYs with similar bitrate	44/48	NA
	1 fast MPHY and 1 slow (1)	43	NA
	1 fast MPHY and 1 slow (2)	41	NA
	1 fast MPHY and 3 slow (3)	38	NA

1. In case of one 25Mbit PHY and 1 XDSL 1Mbit. This example is for N=4 for the first PHY and N=0.16 for the second.
2. In case of one 25Mbit PHY and 1 XDSL 500 Kbit. This example is for N=1 for the first PHY and N=0.02 for the second.
3. In case of one 25Mbit PHY and 3 XDSL 500 Kbit. This example is for N=1 for the first PHY and N=0.02 for the second.
4. APC(N=1)/APC(N=4). In case of N=4, UTOPIA mode, the numbers represent the CP load, but a 100% load cannot be achieved if the CP works solely on transmit. For example if an AAL0 internal channel operates at 30Mbps and APC(N=4) this means that the CP is half loaded. A full CP load that would yield a 60Mbps is not reachable. This is because of the UTOPIA TX implementation which requires the CP to wait for the UTOPIA to finish transmitting a cell before it passes it a new cell.
5. No scrambler / With Scrambler and coset

The following table allows for calculating the impact of several modes, on performance.

Table B-6. Performance Calculation

Mode	Numerator	Denominator (CPM Load in Mbps)
PM on internal channel	Total bit rate of internal channels with PM activated	575
PM on external channel	Total bit rate of external channels with PM activated	457
CRC-10 on AAL0 Tx channel	Total bit rate of CRC-10 AAL0 channels	207
Statistics mode set, single PHY	Total transmitted bit rate	2237
Statistics mode set, multy PHY	Total transmitted bit rate	1342

Example of mixed transmit receive CPM load calculation:

2 transmit external AAL5 channels, each of them is 6Mbps.

One of the two TX AAL5 channels has PM activated.

5 receive, internal channels, address compression, AAL0, each of them is 4Mbps.

MCF is on.

2 Tx AAL5 load = (2*6)/47

PM load = 6/457

5 Rx AAL0 channels = (5*4)/65

MCF load = (5*4)/1678

overall CPM load = 12/47 + 6/457 + 20/65 + 20/1678 = 0.6

Appendix C
Register Quick Reference Guide

This section provides a brief guide to the core registers.

C.1 User Registers

The MPC855T implements the user-level registers defined by the PowerPC architecture except those required for supporting floating-point operations (the floating-point register file (FPRs) and the floating-point status and control register (FPSCR)). User-level registers are listed in Table C-1 and Table C-2. Table C-2 lists user-level special-purpose registers (SPRs).

Table C-1. User-Level Registers

Description	Name	Comments	Access Level	Serialize Access
General-purpose registers	GPRs	The thirty-two 32-bit (GPRs) are used for source and destination operands.	User	—
Condition register	CR	See Section 4.1.1.1.1, “Condition Register (CR).”	User	Only **mtcrf**

Table C-2 lists SPRs defined by the PowerPC architecture implemented on the MPC855T.

Table C-2. User-Level SPRs

SPR Number			Name	Comments	Serialize Access
Decimal	SPR [5–9]	SPR [0–4]			
1	00000	00001	XER	See Section 4.1.1.1.3, “XER.”	Write: Full sync Read: Sync relative to load/store operations
8	00000	01000	LR	See the *Programming Environments Manual*	No
9	00000	01001	CTR	See the *Programming Environments Manual*	No
268	01000	01100	TBL read [1]	Section 10.9, “The Timebase.”	Write (as a store)
269	01000	01101	TBU read [2]		

1 Extended opcode for mftb, 371 rather than 339.
2 Any write (**mtspr**) to this address causes an implementation-dependent software emulation exception.

C.2 Supervisor Registers

All supervisor-level registers implemented on the MPC855T are SPRs, except for the machine state register (MSR), described in Table C-3.

Table C-3. Supervisor-Level Registers

Description	Name	Comments	Serialize Access
Machine state register	MSR	See Section 4.1.2.3.1, “Machine State Register (MSR).”	Write fetch sync

Table C-4 lists supervisor-level SPRs defined by the PowerPC architecture.

Table C-4. Supervisor-Level SPRs

SPR Number			Name	Comments	Serialize Access
Decimal	SPR[5–9]	SPR[0–4]			
18	00000	10010	DSISR	See the *Programming Environments Manual* and Section 4.1.2.1, “DAR, DSISR, and BAR Operation.”	Write: Full sync Read: Sync relative to load/store operations
19	00000	10011	DAR	See the *Programming Environments Manual* and Section 4.1.2.1, “DAR, DSISR, and BAR Operation.”	Write: Full sync Read: Sync relative to load/ store operations
22	00000	10110	DEC	See Section 10.8.1, “Decrementer Register (DEC),” and in Chapter 14, “Clocks and Power Control”	Write
26	00000	11010	SRR0	See SRR0 settings for individual exceptions in Chapter 6, “Exceptions.”	Write
27	00000	11011	SRR1	See SRR1 settings for individual exceptions in Chapter 6, “Exceptions.”	Write
272	01000	10000	SPRG0	See the *Programming Environments Manual*.	Write
273	01000	10001	SPRG1		
274	01000	10010	SPRG2		
275	01000	10011	SPRG3		
284	01000	11100	TBL write [1]	See Section 10.9, “The Timebase,” and Chapter 14, “Clocks and Power Control.”	Write (as a store)
285	01000	11101	TBU write[1]		
287	01000	11111	PVR	Section 4.1.2.3.2, “Processor Version Register.”	No (read-only register)

[1] Any read (**mftb**) to this address causes an implementation-dependent software emulation exception.

C.3 MPC855T-Specific SPRs

Table C-2 and Table C-5 list SPRs specific to the MPC855T. Debug registers, which have additional protection, are described in Chapter 44, “System Development and Debugging.” Supervisor-level registers are described in Table C-5.

Table C-5. MPC855T-Specific Supervisor-Level SPRs

SPR Number			Name	Comments	Serialize Access
Decimal	SPR[5–9]	SPR[0–4]			
80	00010	10000	EIE	See Section 6.1.5, “Recoverability after an Exception.”	Write
81	00010	10001	EID		Write
82	00010	10010	NRI		Write
631	10011	10111	DPIR [1]		Fetch-only
638	10011	11110	IMMR	Section 10.4.1, “Internal Memory Map Register (IMMR).”	Write (as a store)
560	10001	10000	IC_CST	Section 7.3.1, “Instruction Cache Control Registers”	Write (as a store)
561	10001	10001	IC_ADR	Section 7.3.1, “Instruction Cache Control Registers”	Write (as a store)
562	10001	10010	IC_DAT	Section 7.3.1, “Instruction Cache Control Registers”	Write (as a store)
568	10001	11000	DC_CST	Section 7.3.2, “Data Cache Control Registers”	Write (as a store)
569	10001	11001	DC_ADR	Section 7.3.2, “Data Cache Control Registers”	Write (as a store)
570	10001	11010	DC_DAT	Section 7.3.2, “Data Cache Control Registers”	Write (as a store)
784	11000	10000	MI_CTR	Section 8.8.1, “IMMU Control Register (MI_CTR)”	Write (as a store)
786	11000	10010	MI_AP	Section 8.8.10, “MMU Access Protection Registers (MI_AP/MD_AP)”	Write (as a store)
787	11000	10011	MI_EPN	Section 8.8.3, “IMMU/DMMU Effective Page Number Register (Mx_EPN)”	Write (as a store)
789	11000	10101	MI_TWC (MI_L1DL2P)	Section 8.8.4, “IMMU Tablewalk Control Register (MI_TWC)	Write (as a store)
790	11000	10110	MI_RPN	Section 8.8.6, “IMMU Real Page Number Register (MI_RPN)”	Write (as a store)
816	11001	10000	MI_CAM	Section 8.8.12.1, “IMMU CAM Entry Read Register (MI_CAM)”	Write (as a store)
817	11001	10001	MI_RAM0	Section 8.8.12.2, “IMMU RAM Entry Read Register 0 (MI_RAM0)”	Write (as a store)

Table C-5. MPC855T-Specific Supervisor-Level SPRs (continued)

SPR Number			Name	Comments	Serialize Access
Decimal	SPR[5–9]	SPR[0–4]			
818	11001	10010	MI_RAM1	Section 8.8.13, “DMMU RAM Entry Read Register 1 (MD_RAM1)”	Write (as a store)
792	11000	11000	MD_CTR	Section 8.8.2, “DMMU Control Register (MD_CTR).”	Write (as a store)
793	11000	11001	M_CASID	Section 8.8.9, “MMU Current Address Space ID Register (M_CASID)”	Write (as a store)
794	11000	11010	MD_AP	Section 8.8.10, “MMU Access Protection Registers (MI_AP/MD_AP)”	Write (as a store)
795	11000	11011	MD_EPN	Section 8.8.3, “IMMU/DMMU Effective Page Number Register (Mx_EPN)”	Write (as a store)
796	11000	11100	M_TWB (MD_L1P)	Section 8.8.8, “MMU Tablewalk Base Register (M_TWB)”	Write (as a store)
797	11000	11101	MD_TWC (MD_L1DL2P)	Section 8.8.5, “DMMU Tablewalk Control Register (MD_TWC)”	Write (as a store)
798	11000	11110	MD_RPN	Section 8.8.7, “DMMU Real Page Number Register (MD_RPN)”	Write (as a store)
799	11000	11111	M_TW (M_SAVE)	Section 8.8.11, “MMU Tablewalk Special Register (M_TW)”	Write (as a store)
824	11001	11000	MD_CAM	Section 8.8.12.4, “DMMU CAM Entry Read Register (MD_CAM)”	Write (as a store)
825	11001	11001	MD_RAM0	Section 8.8.12.5, “DMMU RAM Entry Read Register 0 (MD_RAM0)”	Write (as a store)
826	11001	11010	MD_RAM1	Section 8.8.13, “DMMU RAM Entry Read Register 1 (MD_RAM1)”	Write (as a store)

[1] Fetch-only register; **mtspr** is ignored; using **mfspr** gives an undefined value.

Debug-level registers are described in Table C-6. These registers are described in Section 44.5.1, “Development Support Registers.”

Table C-6. MPC855T-Specific Debug-Level SPRs

SPR Number			Name	Serialize Access
Decimal	SPR[5–9]	SPR[0–4]		
144	00100	10000	CMPA	Fetch sync on write
145	00100	10001	CMPB	Fetch sync on write
146	00100	10010	CMPC	Fetch sync on write
147	00100	10011	CMPD	Fetch sync on write
148	00100	10100	ICR	Fetch sync on write

Table C-6. MPC855T-Specific Debug-Level SPRs (continued)

SPR Number			Name	Serialize Access
Decimal	SPR[5–9]	SPR[0–4]		
149	00100	10101	DER	Fetch sync on write
150	00100	10110	COUNTA	Fetch sync on write
151	00100	10111	COUNTB	Fetch sync on write
152	00100	11000	CMPE	Write: Fetch sync Read: Sync relative to load/store operations
153	00100	11001	CMPF	Write: Fetch sync Read: Sync relative to load/store operations
154	00100	11010	CMPG	Write: Fetch sync Read: Sync relative to load/store operations
155	00100	11011	CMPH	Write: Fetch sync Read: Sync relative to load/store operations
156	00100	11100	LCTRL1	Write: Fetch sync Read: Sync relative to load/store operations
157	00100	11101	LCTRL2	Write: Fetch sync Read: Sync relative to load/store operations
158	00100	11110	ICTRL	Fetch sync on write
159	00100	11111	BAR	Write: Fetch sync Read: Sync relative to load/store operations. See Section 4.1.2.1, “DAR, DSISR, and BAR Operation.”
630	10011	10110	DPDR	Read and Write

Appendix D
Instruction Set Listings

This appendix lists the MPC855T's instruction set as well as the additional PowerPC instructions not implemented in the MPC855T. Instructions are sorted by mnemonic, opcode, function, and form. Also included in this appendix is a quick reference table that contains general information, such as the architecture level, privilege level, and form, and indicates if the instruction is 64-bit and optional.

Note that split fields, that represent the concatenation of sequences from left to right, are shown in lowercase. For more information refer to Chapter 8, "Instruction Set," in *The Programming Environments Manual.*

D.1 Instructions Sorted by Mnemonic

Table D-1 lists the instructions implemented in the MPC855T's in alphabetical order by mnemonic.

Table D-1. Complete Instruction List Sorted by Mnemonic

Key:

Reserved bits

Instruction not implemented in the MPC855T

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
add*x*	31		D				A					B					OE										Rc
addc*x*	31		D				A					B					OE										Rc
adde*x*	31		D				A					B					OE										Rc
addi	14		D				A					SIMM															
addic	12		D				A					SIMM															
addic.	13		D				A					SIMM															
addis	15		D				A					SIMM															
addme*x*	31		D				A					0 0 0 0 0					OE										Rc
addze*x*	31		D				A					0 0 0 0 0					OE										Rc
and*x*	31		S				A					B															Rc
andc*x*	31		S				A					B															Rc

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
andi.	28	S					A					UIMM															
andis.	29	S					A					UIMM															
b*x*	18	LI																								AA	LK
bc*x*	16	BO					BI					BD														AA	LK
bcctr*x*	19	BO					BI					0 0 0 0 0					528										LK
bclr*x*	19	BO					BI					0 0 0 0 0					16										LK
cmp	31	crfD			0	L	A					B					0										0
cmpi	11	crfD			0	L	A					SIMM															
cmpl	31	crfD			0	L	A					B					32										0
cmpli	10	crfD			0	L	A					UIMM															
cntlzd*x* [4]	31	S					A					0 0 0 0 0					58										Rc
cntlzw*x*	31	S					A					0 0 0 0 0					26										Rc
crand	19	crbD					crbA					crbB					257										0
crandc	19	crbD					crbA					crbB					129										0
creqv	19	crbD					crbA					crbB					289										0
crnand	19	crbD					crbA					crbB					225										0
crnor	19	crbD					crbA					crbB					33										0
cror	19	crbD					crbA					crbB					449										0
crorc	19	crbD					crbA					crbB					417										0
crxor	19	crbD					crbA					crbB					193										0
dcbf	31	0 0 0 0 0					A					B					86										0
dcbi [1]	31	0 0 0 0 0					A					B					470										0
dcbst	31	0 0 0 0 0					A					B					54										0
dcbt	31	0 0 0 0 0					A					B					278										0
dcbtst	31	0 0 0 0 0					A					B					246										0
dcbz	31	0 0 0 0 0					A					B					1014										0
divd*x* [4]	31	D					A					B					OE	489									Rc
divdu*x* [4]	31	D					A					B					OE	457									Rc
divw*x*	31	D					A					B					OE	491									Rc
divwu*x*	31	D					A					B					OE	459									Rc
eciwx	31	D					A					B					310										0
ecowx	31	S					A					B					438										0
eieio	31	0 0 0 0 0					0 0 0 0 0					0 0 0 0 0					854										0
eqv*x*	31	S					A					B					284										Rc

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
extsb*x*	31	S					A					0	0	0	0	0	954										Rc
extsh*x*	31	S					A					0	0	0	0	0	922										Rc
extsw*x* [4]	31	S					A					0	0	0	0	0	986										Rc
fabs*x* [6]	63	D					0	0	0	0	0	B					264										Rc
fadd*x*	63	D					A					B					0	0	0	0	0	21					Rc
fadds*x* [6]	59	D					A					B					0	0	0	0	0	21					Rc
fcfid*x* [4,6]	63	D					0	0	0	0	0	B					846										Rc
fcmpo [6]	63	crfD			0	0	A					B					32										0
fcmpu [6]	63	crfD			0	0	A					B					0										0
fctid*x* [4,6]	63	D					0	0	0	0	0	B					814										Rc
fctidz*x* [4,6]	63	D					0	0	0	0	0	B					815										Rc
fctiw*x* [6]	63	D					0	0	0	0	0	B					14										Rc
fctiwz*x* [6]	63	D					0	0	0	0	0	B					15										Rc
fdiv*x* [6]	63	D					A					B					0	0	0	0	0	18					Rc
fdivs*x* [6]	59	D					A					B					0	0	0	0	0	18					Rc
fmadd*x* [6]	63	D					A					B					C					29					Rc
fmadds*x* [6]	59	D					A					B					C					29					Rc
fmr*x* [6]	63	D					0	0	0	0	0	B					72										Rc
fmsub*x* [6]	63	D					A					B					C					28					Rc
fmsubs*x* [6]	59	D					A					B					C					28					Rc
fmul*x* [6]	63	D					A					0	0	0	0	0	C					25					Rc
fmuls*x* [6]	59	D					A					0	0	0	0	0	C					25					Rc
fnabs*x* [6]	63	D					0	0	0	0	0	B					136										Rc
fneg*x* [6]	63	D					0	0	0	0	0	B					40										Rc
fnmadd*x* [6]	63	D					A					B					C					31					Rc
fnmadds*x* [6]	59	D					A					B					C					31					Rc
fnmsub*x* [6]	63	D					A					B					C					30					Rc
fnmsubs*x* [6]	59	D					A					B					C					30					Rc
fres*x* [5,6]	59	D					0	0	0	0	0	B					0	0	0	0	0	24					Rc
frsp*x* [6]	63	D					0	0	0	0	0	B					12										Rc
frsqrte*x* [5,6]	63	D					0	0	0	0	0	B					0	0	0	0	0	26					Rc
fsel*x* [5,6]	63	D					A					B					C					23					Rc
fsqrt*x* [5,6]	63	D					0	0	0	0	0	B					0	0	0	0	0	22					Rc
fsqrts*x* [5,6]	59	D					0	0	0	0	0	B					0	0	0	0	0	22					Rc

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
fsub*x* [6]	63	D					A					B					0	0	0	0	0	20					Rc
fsubs*x* [6]	59	D					A					B					0	0	0	0	0	20					Rc
icbi	31	0	0	0	0	0	A					B					982										0
isync	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	150										0
lbz	34	D					A					d															
lbzu	35	D					A					d															
lbzux	31	D					A					B					119										0
lbzx	31	D					A					B					87										0
ld [4]	58	D					A					ds															0
ldarx [4]	31	D					A					B					84										0
ldu [4]	58	D					A					ds															1
ldux [4]	31	D					A					B					53										0
ldx [4]	31	D					A					B					21										0
lfd [6]	50	D					A					d															
lfdu [6]	51	D					A					d															
lfdux [6]	31	D					A					B					631										0
lfdx [6]	31	D					A					B					599										0
lfs [6]	48	D					A					d															
lfsu [6]	49	D					A					d															
lfsux [6]	31	D					A					B					567										0
lfsx [6]	31	D					A					B					535										0
lha	42	D					A					d															
lhau	43	D					A					d															
lhaux	31	D					A					B					375										0
lhax	31	D					A					B					343										0
lhbrx	31	D					A					B					790										0
lhz	40	D					A					d															
lhzu	41	D					A					d															
lhzux	31	D					A					B					311										0
lhzx	31	D					A					B					279										0
lmw [3]	46	D					A					d															
lswi [3]	31	D					A					NB					597										0
lswx [3]	31	D					A					B					533										0
lwa [4]	58	D					A					ds															2

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
lwarx	31	D					A					B					20										0
lwaux [4]	31	D					A					B					373										0
lwax [4]	31	D					A					B					341										0
lwbrx	31	D					A					B					534										0
lwz	32	D					A					d														
lwzu	33	D					A					d														
lwzux	31	D					A					B					55										0
lwzx	31	D					A					B					23										0
mcrf	19	crfD			0	0	crfS				0	0	0	0	0	0	0										0
mcrfs [6]	63	crfD			0	0	crfS				0	0	0	0	0	0	64										0
mcrxr	31	crfD			0	0	0	0	0	0	0	0	0	0	0	0	512										0
mfcr	31	D					0	0	0	0	0	0	0	0	0	0	19										0
mffsx [6]	63	D					0	0	0	0	0	0	0	0	0	0	583										Rc
mfmsr [1]	31	D					0	0	0	0	0	0	0	0	0	0	83										0
mfspr [2]	31	D					spr										339										0
mfsr [1]	31	D					0	SR				0	0	0	0	0	595										0
mfsrin [1]	31	D					0	0	0	0	0	B					659										0
mftb	31	D					tbr										371										0
mtcrf	31	S					0	CRM								0	144										0
mtfsb0x [6]	63	crbD					0	0	0	0	0	0	0	0	0	0	70										Rc
mtfsb1x [6]	63	crbD					0	0	0	0	0	0	0	0	0	0	38										Rc
mtfsfx [6]	63	0	FM								0	B					711										Rc
mtfsfix [6]	63	crfD			0	0	0	0	0	0	0	IMM				0	134										Rc
mtmsr [1]	31	S					0	0	0	0	0	0	0	0	0	0	146										0
mtspr [2]	31	S					spr										467										0
mtsr [1]	31	S					0	SR				0	0	0	0	0	210										0
mtsrin [1]	31	S					0	0	0	0	0	B					242										0
mulhdx [4]	31	D					A					B					0	73									Rc
mulhdux [4]	31	D					A					B					0	9									Rc
mulhwx	31	D					A					B					0	75									Rc
mulhwux	31	D					A					B					0	11									Rc
mulldx [4]	31	D					A					B					OE	233									Rc
mulli	7	D					A					SIMM															
mullwx	31	D					A					B					OE	235									Rc

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
nand*x*	31	S					A					B					476										Rc
neg*x*	31	D					A					0 0 0 0 0					OE	104									Rc
nor*x*	31	S					A					B					124										Rc
or*x*	31	S					A					B					444										Rc
orc*x*	31	S					A					B					412										Rc
ori	24	S					A					UIMM															
oris	25	S					A					UIMM															
rfi [1]	19	0 0 0 0 0					0 0 0 0 0					0 0 0 0 0					50										0
rldcl*x* [4]	30	S					A					B					mb						8				Rc
rldcr*x* [4]	30	S					A					B					me						9				Rc
rldic*x* [4]	30	S					A					sh					mb						2			sh	Rc
rldicl*x* [4]	30	S					A					sh					mb						0			sh	Rc
rldicr*x* [4]	30	S					A					sh					me						1			sh	Rc
rldimi*x* [4]	30	S					A					sh					mb						3			sh	Rc
rlwimi*x*	20	S					A					SH					MB					ME					Rc
rlwinm*x*	21	S					A					SH					MB					ME					Rc
rlwnm*x*	23	S					A					B					MB					ME					Rc
sc	17	0 0 0 0 0					0 0 0 0 0					0 0 0 0 0 0 0 0 0 0 0 0 0 0 0													1	0
slbia [1,4,5]	31	0 0 0 0 0					0 0 0 0 0					0 0 0 0 0					498										0
slbie [1,4,5]	31	0 0 0 0 0					0 0 0 0 0					B					434										0
sld*x* [4]	31	S					A					B					27										Rc
slw*x*	31	S					A					B					24										Rc
srad*x* [4]	31	S					A					B					794										Rc
sradi*x* [4]	31	S					A					sh					413									sh	Rc
sraw*x*	31	S					A					B					792										Rc
srawi*x*	31	S					A					SH					824										Rc
srd*x* [4]	31	S					A					B					539										Rc
srw*x*	31	S					A					B					536										Rc
stb	38	S					A					d															
stbu	39	S					A					d															
stbux	31	S					A					B					247										0
stbx	31	S					A					B					215										0
std [4]	62	S					A					ds														0	
stdcx. [4]	31	S					A					B					214										1

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
stdu [4]	62	S					A					ds														1	
stdux [4]	31	S					A					B					181									0	
stdx [4]	31	S					A					B					149									0	
stfd	54	S					A					d															
stfdu	55	S					A					d															
stfdux	31	S					A					B					759									0	
stfdx	31	S					A					B					727									0	
stfiwx [5]	31	S					A					B					983									0	
stfs	52	S					A					d															
stfsu	53	S					A					d															
stfsux	31	S					A					B					695									0	
stfsx	31	S					A					B					663									0	
sth	44	S					A					d															
sthbrx	31	S					A					B					918									0	
sthu	45	S					A					d															
sthux	31	S					A					B					439									0	
sthx	31	S					A					B					407									0	
stmw [3]	47	S					A					d															
stswi [3]	31	S					A					NB					725									0	
stswx [3]	31	S					A					B					661									0	
stw	36	S					A					d															
stwbrx	31	S					A					B					662									0	
stwcx.	31	S					A					B					150									1	
stwu	37	S					A					d															
stwux	31	S					A					B					183									0	
stwx	31	S					A					B					151									0	
subf*x*	31	D					A					B					OE	40								Rc	
subfc*x*	31	D					A					B					OE	8								Rc	
subfe*x*	31	D					A					B					OE	136								Rc	
subfic	08	D					A					SIMM															
subfme*x*	31	D					A					0 0 0 0 0					OE	232								Rc	
subfze*x*	31	D					A					0 0 0 0 0					OE	200								Rc	
sync	31	0 0 0 0 0					0 0 0 0 0					0 0 0 0 0					598									0	
td [4]	31	TO					A					B					68									0	

Name	0	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
tdi [4]	02	TO					A					SIMM															
tlbia [1,5]	31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	7	0							0
tlbie [1,5]	31	0	0	0	0	0	0	0	0	0	0	B					3	0	6								0
tlbsync [1,5]	31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	6	6								0
tw	31	TO					A					B					4										0
twi	03	TO					A					SIMM															
xor*x*	31	S					A					B					3	1	6								Rc
xori	26	S					A					UIMM															
xoris	27	S					A					UIMM															

[1] Supervisor-level instruction
[2] Supervisor- and user-level instruction
[3] Load and store string or multiple instruction
[4] 64-bit instruction
[5] Optional in the PowerPC architecture
[6] Floating-point instructions are not supported by the MPC855T.

D.2 Instructions Sorted by Opcode

Table D-2 lists the instructions defined in the MPC855T in numeric order by opcode.

Key:

Reserved bits

Instruction not implemented in the MPC855T

Table D-2. Complete Instruction List Sorted by Opcode

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
tdi [4]	0 0 0 0 1 0	TO						A					SIMM															
twi	0 0 0 0 1 1	TO						A					SIMM															
mulli	0 0 0 1 1 1	D						A					SIMM															
subfic	0 0 1 0 0 0	D						A					SIMM															
cmpli	0 0 1 0 1 0	crfD			0	L		A					UIMM															
cmpi	0 0 1 0 1 1	crfD			0	L		A					SIMM															
addic	0 0 1 1 0 0	D						A					SIMM															
addic.	0 0 1 1 0 1	D						A					SIMM															
addi	0 0 1 1 1 0	D						A					SIMM															
addis	0 0 1 1 1 1	D						A					SIMM															
bc*x*	0 1 0 0 0 0	BO						BI					BD														AA	LK
sc	0 1 0 0 0 1	0 0 0 0 0						0 0 0 0 0					0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0														1	0
b*x*	0 1 0 0 1 0	LI																									AA	LK
mcrf	0 1 0 0 1 1	crfD			0 0			crfS			0 0		0 0 0 0 0					0 0 0 0 0 0 0 0 0 0										0
bclr*x*	0 1 0 0 1 1	BO						BI					0 0 0 0 0					0 0 0 0 0 1 0 0 0 0										LK
crnor	0 1 0 0 1 1	crbD						crbA					crbB					0 0 0 0 1 0 0 0 0 1										0
rfi	0 1 0 0 1 1	0 0 0 0 0						0 0 0 0 0					0 0 0 0 0					0 0 0 0 1 1 0 0 1 0										0
crandc	0 1 0 0 1 1	crbD						crbA					crbB					0 0 1 0 0 0 0 0 0 1										0
isync	0 1 0 0 1 1	0 0 0 0 0						0 0 0 0 0					0 0 0 0 0					0 0 1 0 0 1 0 1 1 0										0
crxor	0 1 0 0 1 1	crbD						crbA					crbB					0 0 1 1 0 0 0 0 0 1										0
crnand	0 1 0 0 1 1	crbD						crbA					crbB					0 0 1 1 1 0 0 0 0 1										0
crand	0 1 0 0 1 1	crbD						crbA					crbB					0 1 0 0 0 0 0 0 0 1										0
creqv	0 1 0 0 1 1	crbD						crbA					crbB					0 1 0 0 1 0 0 0 0 1										0
crorc	0 1 0 0 1 1	crbD						crbA					crbB					0 1 1 0 1 0 0 0 0 1										0
cror	0 1 0 0 1 1	crbD						crbA					crbB					0 1 1 1 0 0 0 0 0 1										0
bcctr*x*	0 1 0 0 1 1	BO						BI					0 0 0 0 0					1 0 0 0 0 1 0 0 0 0										LK
rlwimi*x*	0 1 0 1 0 0	S						A					SH					MB					ME					Rc
rlwinm*x*	0 1 0 1 0 1	S						A					SH					MB					ME					Rc

Name	0 5	6 7 8	9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
rlwnm*x*	0 1 0 1 1 1	S		A	B	MB	ME	Rc
ori	0 1 1 0 0 0	S		A	UIMM			
oris	0 1 1 0 0 1	S		A	UIMM			
xori	0 1 1 0 1 0	S		A	UIMM			
xoris	0 1 1 0 1 1	S		A	UIMM			
andi.	0 1 1 1 0 0	S		A	UIMM			
andis.	0 1 1 1 0 1	S		A	UIMM			
rldicl*x* [4]	0 1 1 1 1 0	S		A	sh	mb	0 0 0	sh Rc
rldicr*x* [4]	0 1 1 1 1 0	S		A	sh	me	0 0 1	sh Rc
rldic*x* [4]	0 1 1 1 1 0	S		A	sh	mb	0 1 0	sh Rc
rldimi*x* [4]	0 1 1 1 1 0	S		A	sh	mb	0 1 1	sh Rc
rldcl*x* [4]	0 1 1 1 1 0	S		A	B	mb	0 1 0 0 0	Rc
rldcr*x* [4]	0 1 1 1 1 0	S		A	B	me	0 1 0 0 1	Rc
cmp	0 1 1 1 1 1	crfD	0 L	A	B	0 0 0 0 0 0 0 0 0 0		0
tw	0 1 1 1 1 1	TO		A	B	0 0 0 0 0 0 0 1 0 0		0
subfc*x*	0 1 1 1 1 1	D		A	B	OE	0 0 0 0 0 1 0 0 0	Rc
mulhdu*x* [4]	0 1 1 1 1 1	D		A	B	0	0 0 0 0 0 1 0 0 1	Rc
addc*x*	0 1 1 1 1 1	D		A	B	OE	0 0 0 0 0 1 0 1 0	Rc
mulhwu*x*	0 1 1 1 1 1	D		A	B	0	0 0 0 0 0 1 0 1 1	Rc
mfcr	0 1 1 1 1 1	D		0 0 0 0 0	0 0 0 0 0	0 0 0 0 0 1 0 0 1 1		0
lwarx	0 1 1 1 1 1	D		A	B	0 0 0 0 0 1 0 1 0 0		0
ldx [4]	0 1 1 1 1 1	D		A	B	0 0 0 0 0 1 0 1 0 1		0
lwzx	0 1 1 1 1 1	D		A	B	0 0 0 0 0 1 0 1 1 1		0
slw*x*	0 1 1 1 1 1	S		A	B	0 0 0 0 0 1 1 0 0 0		Rc
cntlzw*x*	0 1 1 1 1 1	S		A	0 0 0 0 0	0 0 0 0 0 1 1 0 1 0		Rc
sld*x* [4]	0 1 1 1 1 1	S		A	B	0 0 0 0 0 1 1 0 1 1		Rc
and*x*	0 1 1 1 1 1	S		A	B	0 0 0 0 0 1 1 1 0 0		Rc
cmpl	0 1 1 1 1 1	crfD	0 L	A	B	0 0 0 0 1 0 0 0 0 0		0
subf*x*	0 1 1 1 1 1	D		A	B	OE	0 0 0 1 0 1 0 0 0	Rc
ldux [4]	0 1 1 1 1 1	D		A	B	0 0 0 0 1 1 0 1 0 1		0
dcbst	0 1 1 1 1 1	0 0 0 0 0		A	B	0 0 0 0 1 1 0 1 1 0		0
lwzux	0 1 1 1 1 1	D		A	B	0 0 0 0 1 1 0 1 1 1		0
cntlzd*x* [4]	0 1 1 1 1 1	S		A	0 0 0 0 0	0 0 0 0 1 1 1 0 1 0		Rc
andc*x*	0 1 1 1 1 1	S		A	B	0 0 0 0 1 1 1 1 0 0		Rc

Name	0 5	6 7 8	9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
td [4]	0 1 1 1 1 1	TO	A		B	0 0 0 1 0 0 0 1 0 0	0
mulhd*x* [4]	0 1 1 1 1 1	D	A		B	0 0 1 0 0 1 0 0 1	Rc
mulhw*x*	0 1 1 1 1 1	D	A		B	0 0 1 0 0 1 0 1 1	Rc
mfmsr	0 1 1 1 1 1	D	0 0 0 0 0		0 0 0 0 0	0 0 0 1 0 1 0 0 1 1	0
ldarx [4]	0 1 1 1 1 1	D	A		B	0 0 0 1 0 1 0 1 0 0	0
dcbf	0 1 1 1 1 1	0 0 0 0 0	A		B	0 0 0 1 0 1 0 1 1 0	0
lbzx	0 1 1 1 1 1	D	A		B	0 0 0 1 0 1 0 1 1 1	0
neg*x*	0 1 1 1 1 1	D	A		0 0 0 0 0	OE 0 0 1 1 0 1 0 0 0	Rc
lbzux	0 1 1 1 1 1	D	A		B	0 0 0 1 1 1 0 1 1 1	0
nor*x*	0 1 1 1 1 1	S	A		B	0 0 0 1 1 1 1 1 0 0	Rc
subfe*x*	0 1 1 1 1 1	D	A		B	OE 0 1 0 0 0 1 0 0 0	Rc
adde*x*	0 1 1 1 1 1	D	A		B	OE 0 1 0 0 0 1 0 1 0	Rc
mtcrf	0 1 1 1 1 1	S	0	CRM	0	0 0 1 0 0 1 0 0 0 0	0
mtmsr	0 1 1 1 1 1	S	0 0 0 0 0		0 0 0 0 0	0 0 1 0 0 1 0 0 1 0	0
stdx [4]	0 1 1 1 1 1	S	A		B	0 0 1 0 0 1 0 1 0 1	0
stwcx.	0 1 1 1 1 1	S	A		B	0 0 1 0 0 1 0 1 1 0	1
stwx	0 1 1 1 1 1	S	A		B	0 0 1 0 0 1 0 1 1 1	0
stdux [4]	0 1 1 1 1 1	S	A		B	0 0 1 0 1 1 0 1 0 1	0
stwux	0 1 1 1 1 1	S	A		B	0 0 1 0 1 1 0 1 1 1	0
subfze*x*	0 1 1 1 1 1	D	A		0 0 0 0 0	OE 0 1 1 0 0 1 0 0 0	Rc
addze*x*	0 1 1 1 1 1	D	A		0 0 0 0 0	OE 0 1 1 0 0 1 0 1 0	Rc
mtsr	0 1 1 1 1 1	S	0	SR	0 0 0 0 0	0 0 1 1 0 1 0 0 1 0	0
stdcx. [4]	0 1 1 1 1 1	S	A		B	0 0 1 1 0 1 0 1 1 0	1
stbx	0 1 1 1 1 1	S	A		B	0 0 1 1 0 1 0 1 1 1	0
subfme*x*	0 1 1 1 1 1	D	A		0 0 0 0 0	OE 0 1 1 1 0 1 0 0 0	Rc
mulld [4]	0 1 1 1 1 1	D	A		B	OE 0 1 1 1 0 1 0 0 1	Rc
addme*x*	0 1 1 1 1 1	D	A		0 0 0 0 0	OE 0 1 1 1 0 1 0 1 0	Rc
mullw*x*	0 1 1 1 1 1	D	A		B	OE 0 1 1 1 0 1 0 1 1	Rc
mtsrin	0 1 1 1 1 1	S	0 0 0 0 0		B	0 0 1 1 1 1 0 0 1 0	0
dcbtst	0 1 1 1 1 1	0 0 0 0 0	A		B	0 0 1 1 1 1 0 1 1 0	0
stbux	0 1 1 1 1 1	S	A		B	0 0 1 1 1 1 0 1 1 1	0
add*x*	0 1 1 1 1 1	D	A		B	OE 1 0 0 0 0 1 0 1 0	Rc
dcbt	0 1 1 1 1 1	0 0 0 0 0	A		B	0 1 0 0 0 1 0 1 1 0	0
lhzx	0 1 1 1 1 1	D	A		B	0 1 0 0 0 1 0 1 1 1	0

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
eqv*x*	0 1 1 1 1 1	S	A	B	0 1 0 0 0 1 1 1 0 0	Rc
tlbie [1,5]	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	B	0 1 0 0 1 1 0 0 1 0	0
eciwx	0 1 1 1 1 1	D	A	B	0 1 0 0 1 1 0 1 1 0	0
lhzux	0 1 1 1 1 1	D	A	B	0 1 0 0 1 1 0 1 1 1	0
xor*x*	0 1 1 1 1 1	S	A	B	0 1 0 0 1 1 1 1 0 0	Rc
mfspr [2]	0 1 1 1 1 1	D	spr		0 1 0 1 0 1 0 0 1 1	0
lwax [4]	0 1 1 1 1 1	D	A	B	0 1 0 1 0 1 0 1 0 1	0
lhax	0 1 1 1 1 1	D	A	B	0 1 0 1 0 1 0 1 1 1	0
tlbia [1,5]	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 1 0 1 1 1 0 0 1 0	0
mftb	0 1 1 1 1 1	D	tbr		0 1 0 1 1 1 0 0 1 1	0
lwaux [4]	0 1 1 1 1 1	D	A	B	0 1 0 1 1 1 0 1 0 1	0
lhaux	0 1 1 1 1 1	D	A	B	0 1 0 1 1 1 0 1 1 1	0
sthx	0 1 1 1 1 1	S	A	B	0 1 1 0 0 1 0 1 1 1	0
orc*x*	0 1 1 1 1 1	S	A	B	0 1 1 0 0 1 1 1 0 0	Rc
sradi*x* [4]	0 1 1 1 1 1	S	A	sh	1 1 0 0 1 1 1 0 1 1 sh	Rc
slbie [1,4,5]	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	B	0 1 1 0 1 1 0 0 1 0	0
ecowx	0 1 1 1 1 1	S	A	B	0 1 1 0 1 1 0 1 1 0	0
sthux	0 1 1 1 1 1	S	A	B	0 1 1 0 1 1 0 1 1 1	0
or*x*	0 1 1 1 1 1	S	A	B	0 1 1 0 1 1 1 1 0 0	Rc
divdu*x* [4]	0 1 1 1 1 1	D	A	B	OE 1 1 1 0 0 1 0 0 1	Rc
divwu*x*	0 1 1 1 1 1	D	A	B	OE 1 1 1 0 0 1 0 1 1	Rc
mtspr [2]	0 1 1 1 1 1	S	spr		0 1 1 1 0 1 0 0 1 1	0
dcbi	0 1 1 1 1 1	0 0 0 0 0	A	B	0 1 1 1 0 1 0 1 1 0	0
nand*x*	0 1 1 1 1 1	S	A	B	0 1 1 1 0 1 1 1 0 0	Rc
divd*x* [4]	0 1 1 1 1 1	D	A	B	OE 1 1 1 1 0 1 0 0 1	Rc
divw*x*	0 1 1 1 1 1	D	A	B	OE 1 1 1 1 0 1 0 1 1	Rc
slbia [1,4,5]	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 1 1 1 1 1 0 0 1 0	0
mcrxr	0 1 1 1 1 1	crfD 0 0	0 0 0 0 0	0 0 0 0 0	1 0 0 0 0 0 0 0 0 0	0
lswx [3]	0 1 1 1 1 1	D	A	B	1 0 0 0 0 1 0 1 0 1	0
lwbrx	0 1 1 1 1 1	D	A	B	1 0 0 0 0 1 0 1 1 0	0
lfsx[6]	0 1 1 1 1 1	D	A	B	1 0 0 0 0 1 0 1 1 1	0
srw*x*	0 1 1 1 1 1	S	A	B	1 0 0 0 0 1 1 0 0 0	Rc
srd*x* [4]	0 1 1 1 1 1	S	A	B	1 0 0 0 0 1 1 0 1 1	Rc
tlbsync [1,5]	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	1 0 0 0 1 1 0 1 1 0	0

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lfsux[6]	0 1 1 1 1 1	D	A	B	1 0 0 0 1 1 0 1 1 1	0
mfsr	0 1 1 1 1 1	D	0 SR	0 0 0 0 0	1 0 0 1 0 1 0 0 1 1	0
lswi [3]	0 1 1 1 1 1	D	A	NB	1 0 0 1 0 1 0 1 0 1	0
sync	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	1 0 0 1 0 1 0 1 1 0	0
lfdx[6]	0 1 1 1 1 1	D	A	B	1 0 0 1 0 1 0 1 1 1	0
lfdux[6]	0 1 1 1 1 1	D	A	B	1 0 0 1 1 1 0 1 1 1	0
mfsrin [1]	0 1 1 1 1 1	D	0 0 0 0 0	B	1 0 1 0 0 1 0 0 1 1	0
stswx [3]	0 1 1 1 1 1	S	A	B	1 0 1 0 0 1 0 1 0 1	0
stwbrx	0 1 1 1 1 1	S	A	B	1 0 1 0 0 1 0 1 1 0	0
stfsx	0 1 1 1 1 1	S	A	B	1 0 1 0 0 1 0 1 1 1	0
stfsux	0 1 1 1 1 1	S	A	B	1 0 1 0 1 1 0 1 1 1	0
stswi [3]	0 1 1 1 1 1	S	A	NB	1 0 1 1 0 1 0 1 0 1	0
stfdx[6]	0 1 1 1 1 1	S	A	B	1 0 1 1 0 1 0 1 1 1	0
stfdux[6]	0 1 1 1 1 1	S	A	B	1 0 1 1 1 1 0 1 1 1	0
lhbrx	0 1 1 1 1 1	D	A	B	1 1 0 0 0 1 0 1 1 0	0
sraw*x*	0 1 1 1 1 1	S	A	B	1 1 0 0 0 1 1 0 0 0	Rc
srad*x* [4]	0 1 1 1 1 1	S	A	B	1 1 0 0 0 1 1 0 1 0	Rc
srawi*x*	0 1 1 1 1 1	S	A	SH	1 1 0 0 1 1 1 0 0 0	Rc
eieio	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	1 1 0 1 0 1 0 1 1 0	0
sthbrx	0 1 1 1 1 1	S	A	B	1 1 1 0 0 1 0 1 1 0	0
extsh*x*	0 1 1 1 1 1	S	A	0 0 0 0 0	1 1 1 0 0 1 1 0 1 0	Rc
extsb*x*	0 1 1 1 1 1	S	A	0 0 0 0 0	1 1 1 0 1 1 1 0 1 0	Rc
icbi	0 1 1 1 1 1	0 0 0 0 0	A	B	1 1 1 1 0 1 0 1 1 0	0
stfiwx [5]	0 1 1 1 1 1	S	A	B	1 1 1 1 0 1 0 1 1 1	0
extsw [4]	0 1 1 1 1 1	S	A	0 0 0 0 0	1 1 1 1 0 1 1 0 1 0	Rc
dcbz	0 1 1 1 1 1	0 0 0 0 0	A	B	1 1 1 1 1 1 0 1 1 0	0
lwz	1 0 0 0 0 0	D	A	d		
lwzu	1 0 0 0 0 1	D	A	d		
lbz	1 0 0 0 1 0	D	A	d		
lbzu	1 0 0 0 1 1	D	A	d		
stw	1 0 0 1 0 0	S	A	d		
stwu	1 0 0 1 0 1	S	A	d		
stb	1 0 0 1 1 0	S	A	d		
stbu	1 0 0 1 1 1	S	A	d		

Name	0 5	6 7	8 9	10 11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
lhz	1 0 1 0 0 0	D		A	d			
lhzu	1 0 1 0 0 1	D		A	d			
lha	1 0 1 0 1 0	D		A	d			
lhau	1 0 1 0 1 1	D		A	d			
sth	1 0 1 1 0 0	S		A	d			
sthu	1 0 1 1 0 1	S		A	d			
lmw [3]	1 0 1 1 1 0	D		A	d			
stmw [3]	1 0 1 1 1 1	S		A	d			
lfs[6]	1 1 0 0 0 0	D		A	d			
lfsu[6]	1 1 0 0 0 1	D		A	d			
lfd[6]	1 1 0 0 1 0	D		A	d			
lfdu[6]	1 1 0 0 1 1	D		A	d			
stfs[6]	1 1 0 1 0 0	S		A	d			
stfsu[6]	1 1 0 1 0 1	S		A	d			
stfd[6]	1 1 0 1 1 0	S		A	d			
stfdu[6]	1 1 0 1 1 1	S		A	d			
ld [4]	1 1 1 0 1 0	D		A	ds			0 0
ldu [4]	1 1 1 0 1 0	D		A	ds			0 1
lwa [4]	1 1 1 0 1 0	D		A	ds			1 0
fdivs*x*[6]	1 1 1 0 1 1	D		A	B	0 0 0 0 0	1 0 0 1 0	Rc
fsubs*x*[6]	1 1 1 0 1 1	D		A	B	0 0 0 0 0	1 0 1 0 0	Rc
fadds*x*[6]	1 1 1 0 1 1	D		A	B	0 0 0 0 0	1 0 1 0 1	Rc
fsqrts*x* [5,6]	1 1 1 0 1 1	D		0 0 0 0 0	B	0 0 0 0 0	1 0 1 1 0	Rc
fres*x* [5,6]	1 1 1 0 1 1	D		0 0 0 0 0	B	0 0 0 0 0	1 1 0 0 0	Rc
fmuls*x*[6]	1 1 1 0 1 1	D		A	0 0 0 0 0	C	1 1 0 0 1	Rc
fmsubs*x*[6]	1 1 1 0 1 1	D		A	B	C	1 1 1 0 0	Rc
fmadds*x*[6]	1 1 1 0 1 1	D		A	B	C	1 1 1 0 1	Rc
fnmsubs*x*[6]	1 1 1 0 1 1	D		A	B	C	1 1 1 1 0	Rc
fnmadds*x*[6]	1 1 1 0 1 1	D		A	B	C	1 1 1 1 1	Rc
std [4]	1 1 1 1 1 0	S		A	ds			0 0
stdu [4]	1 1 1 1 1 0	S		A	ds			0 1
fcmpu[6]	1 1 1 1 1 1	crfD	0 0	A	B	0 0 0 0 0 0 0 0 0 0		0
frsp*x*[6]	1 1 1 1 1 1	D		0 0 0 0 0	B	0 0 0 0 0 0 1 1 0 0		Rc
fctiw*x*[6]	1 1 1 1 1 1	D		0 0 0 0 0	B	0 0 0 0 0 0 1 1 1 0		Rc

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
fctiwz*x* [6]	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 0 0 0	0 0 1 1 1	Rc
fdiv*x* [6]	1 1 1 1 1 1	D	A	B	0 0 0 0 0	1 0 0 1 0	Rc
fsub*x* [6]	1 1 1 1 1 1	D	A	B	0 0 0 0 0	1 0 1 0 0	Rc
fadd*x* [6]	1 1 1 1 1 1	D	A	B	0 0 0 0 0	1 0 1 0 1	Rc
fsqrt*x* [5,6]	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 0 0 0	1 0 1 1 0	Rc
fsel*x* [5,6]	1 1 1 1 1 1	D	A	B	C	1 0 1 1 1	Rc
fmul*x* [6]	1 1 1 1 1 1	D	A	0 0 0 0 0	C	1 1 0 0 1	Rc
frsqrte*x* [5,6]	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 0 0 0	1 1 0 1 0	Rc
fmsub*x* [6]	1 1 1 1 1 1	D	A	B	C	1 1 1 0 0	Rc
fmadd*x* [6]	1 1 1 1 1 1	D	A	B	C	1 1 1 0 1	Rc
fnmsub*x* [6]	1 1 1 1 1 1	D	A	B	C	1 1 1 1 0	Rc
fnmadd*x* [6]	1 1 1 1 1 1	D	A	B	C	1 1 1 1 1	Rc
fcmpo [6]	1 1 1 1 1 1	crfD 0 0	A	B	0 0 0 0 1 0 0 0 0 0		0
mtfsb1*x* [6]	1 1 1 1 1 1	crbD	0 0 0 0 0	0 0 0 0 0	0 0 0 0 1 0 0 1 1 0		Rc
fneg*x* [6]	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 0 0 1 0 1 0 0 0		Rc
mcrfs [6]	1 1 1 1 1 1	crfD 0 0	crfS 0 0	0 0 0 0 0	0 0 0 1 0 0 0 0 0 0		0
mtfsb0*x* [6]	1 1 1 1 1 1	crbD	0 0 0 0 0	0 0 0 0 0	0 0 0 1 0 0 0 1 1 0		Rc
fmr*x* [6]	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 0 1 0 0 1 0 0 0		Rc
mtfsfi*x* [6]	1 1 1 1 1 1	crfD 0 0	0 0 0 0 0	IMM 0	0 0 1 0 0 0 0 1 1 0		Rc
fnabs*x* [6]	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 1 0 0 0 1 0 0 0		Rc
fabs*x* [6]	1 1 1 1 1 1	D	0 0 0 0 0	B	0 1 0 0 0 0 1 0 0 0		Rc
mffs*x* [6]	1 1 1 1 1 1	D	0 0 0 0 0	0 0 0 0 0	1 0 0 1 0 0 0 1 1 1		Rc
mtfsf*x* [6]	1 1 1 1 1 1	0 FM 0		B	1 0 1 1 0 0 0 1 1 1		Rc
fctid*x* [4,6]	1 1 1 1 1 1	D	0 0 0 0 0	B	1 1 0 0 1 0 1 1 1 0		Rc
fctidz*x* [4,6]	1 1 1 1 1 1	D	0 0 0 0 0	B	1 1 0 0 1 0 1 1 1 1		Rc
fcfid*x* [4,6]	1 1 1 1 1 1	D	0 0 0 0 0	B	1 1 0 1 0 0 1 1 1 0		Rc

[1] Supervisor-level instruction
[2] Supervisor- and user-level instruction
[3] Load and store string or multiple instruction
[4] 64-bit instruction
[5] Optional in the PowerPC architecture
[6] Floating-point instructions are not supported by the MPC855T.

D.3 Instructions Grouped by Functional Categories

Table D-3 through Table D-30 list the PowerPC instructions defined by the MPC855T grouped by function.

Key:

Reserved bits

Instruction not implemented in the MPC855T

Table D-3. Integer Arithmetic Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
add*x*	31	D	A	B	OE	266	Rc
addc*x*	31	D	A	B	OE	10	Rc
adde*x*	31	D	A	B	OE	138	Rc
addi	14	D	A	SIMM			
addic	12	D	A	SIMM			
addic.	13	D	A	SIMM			
addis	15	D	A	SIMM			
addme*x*	31	D	A	0 0 0 0 0	OE	234	Rc
addze*x*	31	D	A	0 0 0 0 0	OE	202	Rc
divd*x* [4]	31	D	A	B	OE	489	Rc
divdu*x* [4]	31	D	A	B	OE	457	Rc
divw*x*	31	D	A	B	OE	491	Rc
divwu*x*	31	D	A	B	OE	459	Rc
mulhd*x* [4]	31	D	A	B	0	73	Rc
mulhdu*x* [4]	31	D	A	B	0	9	Rc
mulhw*x*	31	D	A	B	0	75	Rc
mulhwu*x*	31	D	A	B	0	11	Rc
mulld [4]	31	D	A	B	OE	233	Rc
mulli	07	D	A	SIMM			
mullw*x*	31	D	A	B	OE	235	Rc
neg*x*	31	D	A	0 0 0 0 0	OE	104	Rc
subf*x*	31	D	A	B	OE	40	Rc
subfc*x*	31	D	A	B	OE	8	Rc
subfic*x*	08	D	A	SIMM			
subfe*x*	31	D	A	B	OE	136	Rc
subfme*x*	31	D	A	0 0 0 0 0	OE	232	Rc
subfze*x*	31	D	A	0 0 0 0 0	OE	200	Rc

Table D-4. Integer Compare Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
cmp	31		crfD			0	L	A					B					0 0 0 0 0 0 0 0 0 0										0
cmpi	11		crfD			0	L	A					SIMM															
cmpl	31		crfD			0	L	A					B					32										0
cmpli	10		crfD			0	L	A					UIMM															

Table D-5. Integer Logical Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
and*x*	31		S					A					B					28										Rc
andc*x*	31		S					A					B					60										Rc
andi.	28		S					A					UIMM															
andis.	29		S					A					UIMM															
cntlzd*x* [4]	31		S					A					0 0 0 0 0					58										Rc
cntlzw*x*	31		S					A					0 0 0 0 0					26										Rc
eqv*x*	31		S					A					B					284										Rc
extsb*x*	31		S					A					0 0 0 0 0					954										Rc
extsh*x*	31		S					A					0 0 0 0 0					922										Rc
extsw*x* [4]	31		S					A					0 0 0 0 0					986										Rc
nand*x*	31		S					A					B					476										Rc
nor*x*	31		S					A					B					124										Rc
or*x*	31		S					A					B					444										Rc
orc*x*	31		S					A					B					412										Rc
ori	24		S					A					UIMM															
oris	25		S					A					UIMM															
xor*x*	31		S					A					B					316										Rc
xori	26		S					A					UIMM															
xoris	27		S					A					UIMM															

Table D-6. Integer Rotate Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
rldcl*x* [4]	30		S					A					B					mb						8				Rc
rldcr*x* [4]	30		S					A					B					me						9				Rc
rldic*x* [4]	30		S					A					sh					mb						2		sh		Rc
rldicl*x* [4]	30		S					A					sh					mb						0		sh		Rc

Table D-6. Integer Rotate Instructions (continued)

rldicr*x* [4]	30	S	A	sh	me	1	sh	Rc
rldimi*x* [4]	30	S	A	sh	mb	3	sh	Rc
rlwimi*x*	22	S	A	SH	MB	ME		Rc
rlwinm*x*	20	S	A	SH	MB	ME		Rc
rlwnm*x*	21	S	A	SH	MB	ME		Rc

Table D-7. Integer Shift Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

sld*x* [4]	31	S	A	B	27	Rc
slw*x*	31	S	A	B	24	Rc
srad*x* [4]	31	S	A	B	794	Rc
sradi*x* [4]	31	S	A	sh	413 sh	Rc
sraw*x*	31	S	A	B	792	Rc
srawi*x*	31	S	A	SH	824	Rc
srd*x* [4]	31	S	A	B	539	Rc
srw*x*	31	S	A	B	536	Rc

Table D-8. Floating-Point Arithmetic Instructions[6]

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

fadd*x*	63	D	A	B	0 0 0 0 0	21	Rc
fadds*x*	59	D	A	B	0 0 0 0 0	21	Rc
fdiv*x*	63	D	A	B	0 0 0 0 0	18	Rc
fdivs*x*	59	D	A	B	0 0 0 0 0	18	Rc
fmul*x*	63	D	A	0 0 0 0 0	C	25	Rc
fmuls*x*	59	D	A	0 0 0 0 0	C	25	Rc
fres*x* [5]	59	D	0 0 0 0 0	B	0 0 0 0 0	24	Rc
frsqrte*x* [5]	63	D	0 0 0 0 0	B	0 0 0 0 0	26	Rc
fsub*x*	63	D	A	B	0 0 0 0 0	20	Rc
fsubs*x*	59	D	A	B	0 0 0 0 0	20	Rc
fsel*x* [5]	63	D	A	B	C	23	Rc
fsqrt*x* [5]	63	D	0 0 0 0 0	B	0 0 0 0 0	22	Rc
fsqrts*x* [5]	59	D	0 0 0 0 0	B	0 0 0 0 0	22	Rc

Table D-9. Floating-Point Multiply-Add Instructions[6]

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
fmadd*x*	63	D	A	B	C	29	Rc
fmadds*x*	59	D	A	B	C	29	Rc
fmsub*x*	63	D	A	B	C	28	Rc
fmsubs*x*	59	D	A	B	C	28	Rc
fnmadd*x*	63	D	A	B	C	31	Rc
fnmadds*x*	59	D	A	B	C	31	Rc
fnmsub*x*	63	D	A	B	C	30	Rc
fnmsubs*x*	59	D	A	B	C	30	Rc

Table D-10. Floating-Point Rounding and Conversion Instructions[6]

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
fcfid*x* [4]	63	D	0 0 0 0 0	B	846		Rc
fctid*x* [4]	63	D	0 0 0 0 0	B	814		Rc
fctidz*x* [4]	63	D	0 0 0 0 0	B	815		Rc
fctiw*x*	63	D	0 0 0 0 0	B	14		Rc
fctiwz*x*	63	D	0 0 0 0 0	B	15		Rc
frsp*x*	63	D	0 0 0 0 0	B	12		Rc

Table D-11. Floating-Point Compare Instructions[6]

Name	0 5	6 7 8	9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
fcmpo	63	crfD	0 0	A	B	32		0
fcmpu	63	crfD	0 0	A	B	0		0

Table D-12. Floating-Point Status and Control Register Instructions[6]

Name	0 5	6 7 8	9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
mcrfs	63	crfD	0 0	crfS	0 0 0 0 0	64		0
mffs*x*	63	D		0 0 0 0 0	0 0 0 0 0	583		Rc
mtfsb0*x*	63	crbD		0 0 0 0 0	0 0 0 0 0	70		Rc
mtfsb1*x*	63	crbD		0 0 0 0 0	0 0 0 0 0	38		Rc
mtfsf*x*	63	0	FM	0	B	711		Rc
mtfsfi*x*	63	crfD	0 0	0 0 0 0 0	IMM 0	134		Rc

Table D-13. Integer Load Instructions

Name	0 5	6 7 8	9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lbz	34	D	A	d			
lbzu	35	D	A	d			
lbzux	31	D	A	B	119		0
lbzx	31	D	A	B	87		0
ld [4]	58	D	A	ds			0
ldu [4]	58	D	A	ds			1
ldux [4]	31	D	A	B	53		0
ldx [4]	31	D	A	B	21		0
lha	42	D	A	d			
lhau	43	D	A	d			
lhaux	31	D	A	B	375		0
lhax	31	D	A	B	343		0
lhz	40	D	A	d			
lhzu	41	D	A	d			
lhzux	31	D	A	B	311		0
lhzx	31	D	A	B	279		0
lwa [4]	58	D	A	ds			2
lwaux [4]	31	D	A	B	373		0
lwax [4]	31	D	A	B	341		0
lwz	32	D	A	d			
lwzu	33	D	A	d			
lwzux	31	D	A	B	55		0
lwzx	31	D	A	B	23		0

Table D-14. Integer Store Instructions

Name	0 5	6 7 8	9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
stb	38	S	A	d			
stbu	39	S	A	d			
stbux	31	S	A	B	247		0
stbx	31	S	A	B	215		0
std [4]	62	S	A	ds			0
stdu [4]	62	S	A	ds			1
stdux [4]	31	S	A	B	181		0

Table D-14. Integer Store Instructions (continued)

	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
stdx [4]	31	S						A					B					149										0
sth	44	S						A					d															
sthu	45	S						A					d															
sthux	31	S						A					B					439										0
sthx	31	S						A					B					407										0
stw	36	S						A					d															
stwu	37	S						A					d															
stwux	31	S						A					B					183										0
stwx	31	S						A					B					151										0

Table D-15. Integer Load and Store with Byte-Reverse Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
lhbrx	31	D						A					B					790										0
lwbrx	31	D						A					B					534										0
sthbrx	31	S						A					B					918										0
stwbrx	31	S						A					B					662										0

Table D-16. Integer Load and Store Multiple Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
lmw [3]	46	D						A					d															
stmw [3]	47	S						A					d															

Table D-17. Integer Load and Store String Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
lswi [3]	31	D						A					NB					597										0
lswx [3]	31	D						A					B					533										0
stswi [3]	31	S						A					NB					725										0
stswx [3]	31	S						A					B					661										0

Table D-18. Memory Synchronization Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
eieio	31	0 0 0 0 0						0 0 0 0 0					0 0 0 0 0					854										0
isync	19	0 0 0 0 0						0 0 0 0 0					0 0 0 0 0					150										0
ldarx [4]	31	D						A					B					84										0
lwarx	31	D						A					B					20										0

Table D-18. Memory Synchronization Instructions

stdcx.[4]	31	S	A	B	214	1
stwcx.	31	S	A	B	150	1
sync	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	598	0

Table D-19. Floating-Point Load Instructions[6]

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
lfd	50	D						A					d															
lfdu	51	D						A					d															
lfdux	31	D						A					B					631										0
lfdx	31	D						A					B					599										0
lfs	48	D						A					d															
lfsu	49	D						A					d															
lfsux	31	D						A					B					567										0
lfsx	31	D						A					B					535										0

Table D-20. Floating-Point Store Instructions[6]

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
stfd	54	S						A					d															
stfdu	55	S						A					d															
stfdux	31	S						A					B					759										0
stfdx	31	S						A					B					727										0
stfiwx [5]	31	S						A					B					983										0
stfs	52	S						A					d															
stfsu	53	S						A					d															
stfsux	31	S						A					B					695										0
stfsx	31	S						A					B					663										0

Table D-21. Floating-Point Move Instructions[6]

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
fabs*x*	63	D						0 0 0 0 0					B					264										Rc
fmr*x*	63	D						0 0 0 0 0					B					72										Rc
fnabs*x*	63	D						0 0 0 0 0					B					136										Rc
fneg*x*	63	D						0 0 0 0 0					B					40										Rc

Table D-22. Branch Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
b*x*	18	LI																									AA	LK
bc*x*	16	BO					BI						BD														AA	LK
bcctr*x*	19	BO					BI						0 0 0 0 0					528										LK
bclr*x*	19	BO					BI						0 0 0 0 0					16										LK

Table D-23. Condition Register Logical Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
crand	19	crbD						crbA					crbB					257										0
crandc	19	crbD						crbA					crbB					129										0
creqv	19	crbD						crbA					crbB					289										0
crnand	19	crbD						crbA					crbB					225										0
crnor	19	crbD						crbA					crbB					33										0
cror	19	crbD						crbA					crbB					449										0
crorc	19	crbD						crbA					crbB					417										0
crxor	19	crbD						crbA					crbB					193										0
mcrf	19	crfD				0 0		crfS				0 0	0 0 0 0 0					0 0 0 0 0 0 0 0 0 0										0

Table D-24. System Linkage Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
rfi [1]	19	0 0 0 0 0						0 0 0 0 0					0 0 0 0 0					50										0
sc	17	0 0 0 0 0						0 0 0 0 0					0 0 0 0 0 0 0 0 0 0 0 0 0 0 0														1	0

Table D-25. Trap Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
td [4]	31	TO						A					B					68										0
tdi [4]	03	TO						A					SIMM															
tw	31	TO						A					B					4										0
twi	03	TO						A					SIMM															

Table D-26. Processor Control Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
mcrxr	31	crfS				0 0		0 0 0 0 0					0 0 0 0 0					512										0
mfcr	31	D						0 0 0 0 0					0 0 0 0 0					19										0

Table D-26. Processor Control Instructions

mfmsr [1]	31	D	0 0 0 0 0	0 0 0 0 0	83	0
mfspr [2]	31	D	spr		339	0
mftb	31	D	tpr		371	0
mtcrf	31	S	0	CRM 0	144	0
mtmsr [1]	31	S	0 0 0 0 0	0 0 0 0 0	146	0
mtspr [2]	31	D	spr		467	0

Table D-27. Cache Management Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31		
dcbf	31		0	0	0	0	0	A					B					86											0
dcbi [1]	31		0	0	0	0	0	A					B					470											0
dcbst	31		0	0	0	0	0	A					B					54											0
dcbt	31		0	0	0	0	0	A					B					278											0
dcbtst	31		0	0	0	0	0	A					B					246											0
dcbz	31		0	0	0	0	0	A					B					1014											0
icbi	31		0	0	0	0	0	A					B					982											0

Table D-28. Segment Register Manipulation Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31		
mfsr [1]	31		D					0	SR				0 0 0 0 0					595											0
mfsrin [1]	31		D					0 0 0 0 0					B					659											0
mtsr [1]	31		S					0	SR				0 0 0 0 0					210											0
mtsrin [1]	31		S					0 0 0 0 0					B					242											0

Table D-29. Lookaside Buffer Management Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31		
slbia [1,4,5]	31		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	498											0
slbie [1,4,5]	31		0	0	0	0	0	0	0	0	0	0	B					434											0
tlbia [1,5]	31		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	370											0
tlbie [1,5]	31		0	0	0	0	0	0	0	0	0	0	B					306											0
tlbsync [1,5]	31		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	566											0

Table D-30. External Control Instructions

Name	0 – 5	6 – 7	8 – 9	10 – 11	12 – 13	14 – 15	16 – 17	18 – 19	20 – 21	22 – 23	24 – 25	26 – 27	28 – 29	30	31
eciwx	31	D		A		B		310							0
ecowx	31	S		A		B		438							0

[1] Supervisor-level instruction
[2] Supervisor- and user-level instruction
[3] Load and store string or multiple instruction
[4] 64-bit instruction
[5] Optional in the PowerPC architecture
[6] Floating-point instructions are not supported by the MPC855T.

D.4 Instructions Sorted by Form

Table D-31 through Table D-45 list the PowerPC instructions defined by the MPC855T grouped by form.

Key:

Reserved bits

Instruction not implemented in the MPC855T

Table D-31. I-Form

OPCD	LI	AA	LK

Specific Instruction

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
b*x*	18	LI																									AA	LK

Table D-32. B-Form

OPCD	BO	BI	BD	AA	LK

Specific Instruction

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
bc*x*	16	BO					BI						BD														AA	LK

Table D-33. SC-Form

OPCD	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0

Specific Instruction

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
sc	17	0 0 0 0 0					0 0 0 0 0						0 0 0 0 0 0 0 0 0 0 0 0 0 0 0														1	0

Table D-34. D-Form

OPCD	D	A	d
OPCD	D	A	SIMM
OPCD	S	A	d
OPCD	S	A	UIMM
OPCD	crfD 0 L	A	SIMM
OPCD	crfD 0 L	A	UIMM
OPCD	TO	A	SIMM

Specific Instructions

Name	0 5	6 7 8	9	10	11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
addi	14	D			A	SIMM
addic	12	D			A	SIMM
addic.	13	D			A	SIMM
addis	15	D			A	SIMM
andi.	28	S			A	UIMM
andis.	29	S			A	UIMM
cmpi	11	crfD	0	L	A	SIMM
cmpli	10	crfD	0	L	A	UIMM
lbz	34	D			A	d
lbzu	35	D			A	d
lfd[6]	50	D			A	d
lfdu [6]	51	D			A	d
lfs[6]	48	D			A	d
lfsu[6]	49	D			A	d
lha	42	D			A	d
lhau	43	D			A	d
lhz	40	D			A	d
lhzu	41	D			A	d
lmw [3]	46	D			A	d
lwz	32	D			A	d
lwzu	33	D			A	d
mulli	7	D			A	SIMM
ori	24	S			A	UIMM
oris	25	S			A	UIMM
stb	38	S			A	d
stbu	39	S			A	d
stfd[6]	54	S			A	d
stfdu[6]	55	S			A	d
stfs[6]	52	S			A	d
stfsu[6]	53	S			A	d
sth	44	S			A	d
sthu	45	S			A	d
stmw [3]	47	S			A	d

stw	36	S	A	d
stwu	37	S	A	d
subfic	08	D	A	SIMM
tdi [4]	02	TO	A	SIMM
twi	03	TO	A	SIMM
xori	26	S	A	UIMM
xoris	27	S	A	UIMM

Table D-35. DS-Form

OPCD	D	A	ds	XO
OPCD	S	A	ds	XO

Specific Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
ld [4]	58	D	A	ds			0
ldu [4]	58	D	A	ds			1
lwa [4]	58	D	A	ds			2
std [4]	62	S	A	ds			0
stdu [4]	62	S	A	ds			1

Table D-36. X-Form

OPCD	D	A	B	XO	0
OPCD	D	A	NB	XO	0
OPCD	D	0 0 0 0 0	B	XO	0
OPCD	D	0 0 0 0 0	0 0 0 0 0	XO	0
OPCD	D	0 SR	0 0 0 0 0	XO	0
OPCD	S	A	B	XO	Rc
OPCD	S	A	B	XO	1
OPCD	S	A	B	XO	0
OPCD	S	A	NB	XO	0
OPCD	S	A	0 0 0 0 0	XO	Rc
OPCD	S	0 0 0 0 0	B	XO	0
OPCD	S	0 0 0 0 0	0 0 0 0 0	XO	0
OPCD	S	0 SR	0 0 0 0 0	XO	0
OPCD	S	A	SH	XO	Rc
OPCD	crfD 0 L	A	B	XO	0

OPCD	crfD	0 0	A	B	XO	0
OPCD	crfD	0 0	crfS 0 0	0 0 0 0 0	XO	0
OPCD	crfD	0 0	0 0 0 0 0	0 0 0 0 0	XO	0
OPCD	crfD	0 0	0 0 0 0 0	IMM 0	XO	Rc
OPCD	TO		A	B	XO	0
OPCD	D		0 0 0 0 0	B	XO	Rc
OPCD	D		0 0 0 0 0	0 0 0 0 0	XO	Rc
OPCD	crbD		0 0 0 0 0	0 0 0 0 0	XO	Rc
OPCD	0 0 0 0 0		A	B	XO	0
OPCD	0 0 0 0 0		0 0 0 0 0	B	XO	0
OPCD	0 0 0 0 0		0 0 0 0 0	0 0 0 0 0	XO	0

Specific Instructions

and*x*	31	S	A	B	28	Rc
andc*x*	31	S	A	B	60	Rc
cmp	31	crfD 0 L	A	B	0	0
cmpl	31	crfD 0 L	A	B	32	0
cntlzd*x* [4]	31	S	A	0 0 0 0 0	58	Rc
cntlzw*x*	31	S	A	0 0 0 0 0	26	Rc
dcbf	31	0 0 0 0 0	A	B	86	0
dcbi [1]	31	0 0 0 0 0	A	B	470	0
dcbst	31	0 0 0 0 0	A	B	54	0
dcbt	31	0 0 0 0 0	A	B	278	0
dcbtst	31	0 0 0 0 0	A	B	246	0
dcbz	31	0 0 0 0 0	A	B	1014	0
eciwx	31	D	A	B	310	0
ecowx	31	S	A	B	438	0
eieio	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	854	0
eqv*x*	31	S	A	B	284	Rc
extsb*x*	31	S	A	0 0 0 0 0	954	Rc
extsh*x*	31	S	A	0 0 0 0 0	922	Rc
extsw*x* [4]	31	S	A	0 0 0 0 0	986	Rc
fabs*x* [6]	63	D	0 0 0 0 0	B	264	Rc
fcfid*x* [4,6]	63	D	0 0 0 0 0	B	846	Rc
fcmpo [6]	63	crfD 0 0	A	B	32	0
fcmpu [6]	63	crfD 0 0	A	B	0	0

fctid*x* [4,6]	63	D	0 0 0 0 0	B	814	Rc
fctidz*x* [4,6]	63	D	0 0 0 0 0	B	815	Rc
fctiw*x* [6]	63	D	0 0 0 0 0	B	14	Rc
fctiwz*x* [6]	63	D	0 0 0 0 0	B	15	Rc
fmr*x* [6]	63	D	0 0 0 0 0	B	72	Rc
fnabs*x* [6]	63	D	0 0 0 0 0	B	136	Rc
fneg*x* [6]	63	D	0 0 0 0 0	B	40	Rc
frsp*x* [6]	63	D	0 0 0 0 0	B	12	Rc
icbi	31	0 0 0 0 0	A	B	982	0
lbzux	31	D	A	B	119	0
lbzx	31	D	A	B	87	0
ldarx [4]	31	D	A	B	84	0
ldux [4]	31	D	A	B	53	0
ldx [4]	31	D	A	B	21	0
lfdux [6]	31	D	A	B	631	0
lfdx [6]	31	D	A	B	599	0
lfsux [6]	31	D	A	B	567	0
lfsx [6]	31	D	A	B	535	0
lhaux	31	D	A	B	375	0
lhax	31	D	A	B	343	0
lhbrx	31	D	A	B	790	0
lhzux	31	D	A	B	311	0
lhzx	31	D	A	B	279	0
lswi [3]	31	D	A	NB	597	0
lswx [3]	31	D	A	B	533	0
lwarx	31	D	A	B	20	0
lwaux [4]	31	D	A	B	373	0
lwax [4]	31	D	A	B	341	0
lwbrx	31	D	A	B	534	0
lwzux	31	D	A	B	55	0
lwzx	31	D	A	B	23	0
mcrfs	63	crfD 0 0	crfS 0 0	0 0 0 0 0	64	0
mcrxr	31	crfD 0 0	0 0 0 0 0	0 0 0 0 0	512	0
mfcr	31	D	0 0 0 0 0	0 0 0 0 0	19	0
mffs*x* [6]	63	D	0 0 0 0 0	0 0 0 0 0	583	Rc
mfmsr [1]	31	D	0 0 0 0 0	0 0 0 0 0	83	0

mfsr [1]	31	D	0 SR	0 0 0 0 0	595	0
mfsrin [1]	31	D	0 0 0 0 0	B	659	0
mtfsb0*x* [6]	63	crbD	0 0 0 0 0	0 0 0 0 0	70	Rc
mtfsb1*x* [6]	63	crfD	0 0 0 0 0	0 0 0 0 0	38	Rc
mtfsfi*x* [6]	63	crbD 0 0	0 0 0 0 0	IMM 0	134	Rc
mtmsr [1]	31	S	0 0 0 0 0	0 0 0 0 0	146	0
mtsr [1]	31	S	0 SR	0 0 0 0 0	210	0
mtsrin [1]	31	S	0 0 0 0 0	B	242	0
nand*x*	31	S	A	B	476	Rc
nor*x*	31	S	A	B	124	Rc
or*x*	31	S	A	B	444	Rc
orc*x*	31	S	A	B	412	Rc
slbia [1,4,5]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	498	0
slbie [1,4,5]	31	0 0 0 0 0	0 0 0 0 0	B	434	0
sld*x* [4]	31	S	A	B	27	Rc
slw*x*	31	S	A	B	24	Rc
srad*x* [4]	31	S	A	B	794	Rc
sraw*x*	31	S	A	B	792	Rc
srawi*x*	31	S	A	SH	824	Rc
srd*x* [4]	31	S	A	B	539	Rc
srw*x*	31	S	A	B	536	Rc
stbux	31	S	A	B	247	0
stbx	31	S	A	B	215	0
stdcx. [4]	31	S	A	B	214	1
stdux [4]	31	S	A	B	181	0
stdx [4]	31	S	A	B	149	0
stfdux [6]	31	S	A	B	759	0
stfdx [6]	31	S	A	B	727	0
stfiwx [5,6]	31	S	A	B	983	0
stfsux [6]	31	S	A	B	695	0
stfsx [6]	31	S	A	B	663	0
sthbrx	31	S	A	B	918	0
sthux	31	S	A	B	439	0
sthx	31	S	A	B	407	0
stswi [3]	31	S	A	NB	725	0
stswx [3]	31	S	A	B	661	0

stwbrx	31	S	A	B	662	0
stwcx.	31	S	A	B	150	1
stwux	31	S	A	B	183	0
stwx	31	S	A	B	151	0
sync	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	598	0
td [4]	31	TO	A	B	68	0
tlbia [1,5]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	370	0
tlbie [1,5]	31	0 0 0 0 0	0 0 0 0 0	B	306	0
tlbsync[1,5]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	566	0
tw	31	TO	A	B	4	0
xor*x*	31	S	A	B	316	Rc

Table D-37. XL-Form

OPCD	BO	BI	0 0 0 0 0	XO	LK
OPCD	crbD	crbA	crbB	XO	0
OPCD	crfD 0 0	crfS 0 0	0 0 0 0 0	XO	0
OPCD	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	XO	0

Specific Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
bcctr*x*	19	BO	BI	0 0 0 0 0	528	LK
bclr*x*	19	BO	BI	0 0 0 0 0	16	LK
crand	19	crbD	crbA	crbB	257	0
crandc	19	crbD	crbA	crbB	129	0
creqv	19	crbD	crbA	crbB	289	0
crnand	19	crbD	crbA	crbB	225	0
crnor	19	crbD	crbA	crbB	33	0
cror	19	crbD	crbA	crbB	449	0
crorc	19	crbD	crbA	crbB	417	0
crxor	19	crbD	crbA	crbB	193	0
isync	19	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	150	0
mcrf	19	crfD 0 0	crfS 0 0	0 0 0 0 0	0	0
rfi [1]	19	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	50	0

Table D-38. XFX-Form

OPCD	D	spr	XO	0
OPCD	D	0 CRM 0	XO	0
OPCD	S	spr	XO	0
OPCD	D	tbr	XO	0

Specific Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
mfspr [2]	31		D					spr										339										0
mftb	31		D					tbr										371										0
mtcrf	31		S				0	CRM					0					144										0
mtspr [2]	31		D					spr										467										0

Table D-39. XFL-Form

OPCD	0	FM	0	B	XO	Rc

Specific Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
mtfsf*x* [6]	63		0					FM					0					B										Rc

Table D-40. XS-Form

OPCD	S	A	sh	XO	sh	Rc

Specific Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
sradi*x* [4]	31			S				A					sh					413									sh	Rc

Table D-41. XO-Form

OPCD	D	A	B	OE	XO	Rc
OPCD	D	A	B	0	XO	Rc
OPCD	D	A	0 0 0 0 0	OE	XO	Rc

Specific Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
add*x*	31			D				A					B					OE						266				Rc
addc*x*	31			D				A					B					OE						10				Rc
adde*x*	31			D				A					B					OE						138				Rc

addme*x*	31	D	A	0 0 0 0 0	OE	234	Rc
addze*x*	31	D	A	0 0 0 0 0	OE	202	Rc
divd*x* [4]	31	D	A	B	OE	489	Rc
divdu*x* [4]	31	D	A	B	OE	457	Rc
divw*x*	31	D	A	B	OE	491	Rc
divwu*x*	31	D	A	B	OE	459	Rc
mulhd*x* [4]	31	D	A	B	0	73	Rc
mulhdu*x* [4]	31	D	A	B	0	9	Rc
mulhw*x*	31	D	A	B	0	75	Rc
mulhwu*x*	31	D	A	B	0	11	Rc
mulld*x* [4]	31	D	A	B	OE	233	Rc
mullw*x*	31	D	A	B	OE	235	Rc
neg*x*	31	D	A	0 0 0 0 0	OE	104	Rc
subf*x*	31	D	A	B	OE	40	Rc
subfc*x*	31	D	A	B	OE	8	Rc
subfe*x*	31	D	A	B	OE	136	Rc
subfme*x*	31	D	A	0 0 0 0 0	OE	232	Rc
subfze*x*	31	D	A	0 0 0 0 0	OE	200	Rc

Table D-42. A-Form

OPCD	D	A	B	0 0 0 0 0	XO	Rc
OPCD	D	A	B	C	XO	Rc
OPCD	D	A	0 0 0 0 0	C	XO	Rc
OPCD	D	0 0 0 0 0	B	0 0 0 0 0	XO	Rc

Specific Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

fadd*x* [6]	63	D	A	B	0 0 0 0 0	21	Rc
fadds*x* [6]	59	D	A	B	0 0 0 0 0	21	Rc
fdiv*x* [6]	63	D	A	B	0 0 0 0 0	18	Rc
fdivs*x* [6]	59	D	A	B	0 0 0 0 0	18	Rc
fmadd*x* [6]	63	D	A	B	C	29	Rc
fmadds*x* [6]	59	D	A	B	C	29	Rc
fmsub*x* [6]	63	D	A	B	C	28	Rc
fmsubs*x* [6]	59	D	A	B	C	28	Rc
fmul*x* [6]	63	D	A	0 0 0 0 0	C	25	Rc

fmuls*x*[6]	59	D	A	0 0 0 0 0	C	25	Rc
fnmadd*x* [6]	63	D	A	B	C	31	Rc
fnmadds*x*[6]	59	D	A	B	C	31	Rc
fnmsub*x*[6]	63	D	A	B	C	30	Rc
fnmsubs*x*[6]	59	D	A	B	C	30	Rc
fres*x* [5,6]	59	D	0 0 0 0 0	B	0 0 0 0 0	24	Rc
frsqrte*x* [5,6]	63	D	0 0 0 0 0	B	0 0 0 0 0	26	Rc
fsel*x* [5,6]	63	D	A	B	C	23	Rc
fsqrt*x* [5,6]	63	D	0 0 0 0 0	B	0 0 0 0 0	22	Rc
fsqrts*x* [5,6]	59	D	0 0 0 0 0	B	0 0 0 0 0	22	Rc
fsub*x*[6]	63	D	A	B	0 0 0 0 0	20	Rc
fsubs*x*[6]	59	D	A	B	0 0 0 0 0	20	Rc

Table D-43. M-Form

OPCD	S	A	SH	MB	ME	Rc
OPCD	S	A	B	MB	ME	Rc

Specific Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
rlwimi*x*	20	S					A						SH					MB					ME				Rc
rlwinm*x*	21	S					A						SH					MB					ME				Rc
rlwnm*x*	23	S					A						B					MB					ME				Rc

Table D-44. MD-Form

OPCD	S	A	sh	mb	XO	sh	Rc
OPCD	S	A	sh	me	XO	sh	Rc

Specific Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
rldic*x* [4]	30	S					A						sh					mb					2			sh	Rc
rldicl*x* [4]	30	S					A						sh					mb					0			sh	Rc
rldicr*x* [4]	30	S					A						sh					me					1			sh	Rc
rldimi*x* [4]	30	S					A						sh					mb					3			sh	Rc

Table D-45. MDS-Form

OPCD	S	A	B	mb	XO	Rc
OPCD	S	A	B	me	XO	Rc

Specific Instructions

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
rldcl*x* [4]	30		S					A					B					mb						8				Rc
rldcr*x* [4]	30		S					A					B					me						9				Rc

[1] Supervisor-level instruction
[2] Supervisor- and user-level instruction
[3] Load and store string or multiple instruction
[4] 64-bit instruction
[5] Optional in the PowerPC architecture
[6] Floating-point instructions are not supported by the MPC855T

D.5 Instruction Set Legend

Table D-46 provides general information on the PowerPC instruction set defined by the MPC855T (such as the architectural level, privilege level, and form).

Key:

Reserved bits

Instruction not implemented in the MPC855T

Table D-46. Instruction Set Legend

	UISA	VEA	OEA	Supervisor Level	64-Bit	Optional	Form
add*x*	√						XO
addc*x*	√						XO
adde*x*	√						XO
addi	√						D
addic	√						D
addic.	√						D
addis	√						D
addme*x*	√						XO
addze*x*	√						XO
and*x*	√						X
andc*x*	√						X
andi.	√						D
andis.	√						D
b*x*	√						I
bc*x*	√						B
bcctr*x*	√						XL
bclr*x*	√						XL
cmp	√						X
cmpi	√						D
cmpl	√						X
cmpli	√						D
cntlzd*x* [4]	√				√		X
cntlzw*x*	√						X
crand	√						XL
crandc	√						XL
creqv	√						XL
crnand	√						XL

	UISA	VEA	OEA	Supervisor Level	64-Bit	Optional	Form
crnor	√						XL
cror	√						XL
crorc	√						XL
crxor	√						XL
dcbf		√					X
dcbi [1]			√	√			X
dcbst		√					X
dcbt		√					X
dcbtst		√					X
dcbz		√					X
divd*x* [4]	√				√		XO
divdu*x* [4]	√				√		XO
divw*x*	√						XO
divwu*x*	√						XO
eciwx		√				√	X
ecowx		√				√	X
eieio		√					X
eqv*x*	√						X
extsb*x*	√						X
extsh*x*	√						X
extsw*x* [4]	√				√		X
fabs*x* [6]	√						X
fadd*x* [6]	√						A
fadds*x* [6]	√						A
fcfid*x* [4,6]	√				√		X
fcmpo [6]	√						X
fcmpu [6]	√						X
fctid*x* [4,6]	√				√		X
fctidz*x* [4,6]	√				√		X
fctiw*x* [6]	√						X
fctiwz*x* [6]	√						X
fdiv*x* [6]	√						A
fdivs*x* [6]	√						A
fmadd*x* [6]	√						A

	UISA	VEA	OEA	Supervisor Level	64-Bit	Optional	Form
fmadds*x* [6]	√						A
fmr*x* [6]	√						X
fmsub*x* [6]	√						A
fmsubs*x* [6]	√						A
fmul*x* [6]	√						A
fmuls*x* [6]	√						A
fnabs*x* [6]	√						X
fneg*x* [6]	√						X
fnmadd*x* [6]	√						A
fnmadds*x* [6]	√						A
fnmsub*x* [6]	√						A
fnmsubs*x* [6]	√						A
fres*x* [5,6]	√					√	A
frsp*x* [6]	√						X
frsqrte*x* [5,6]	√					√	A
fsel*x* [5,6]	√					√	A
fsqrt*x* [5,6]	√					√	A
fsqrts*x* [5,6]	√					√	A
fsub*x* [6]	√						A
fsubs*x* [6]	√						A
icbi		√					X
isync		√					XL
lbz	√						D
lbzu	√						D
lbzux	√						X
lbzx	√						X
ld [4]	√				√		DS
ldarx [4]	√				√		X
ldu [4]	√				√		DS
ldux [4]	√				√		X
ldx [4]	√				√		X
lfd [6]	√						D
lfdu [6]	√						D
lfdux [6]	√						X

	UISA	VEA	OEA	Supervisor Level	64-Bit	Optional	Form
lfdx[6]	√						X
lfs[6]	√						D
lfsu[6]	√						D
lfsux[6]	√						X
lfsx[6]	√						X
lha	√						D
lhau	√						D
lhaux	√						X
lhax	√						X
lhbrx	√						X
lhz	√						D
lhzu	√						D
lhzux	√						X
lhzx	√						X
lmw [3]	√						D
lswi [3]	√						X
lswx [3]	√						X
lwa [4]	√				√		DS
lwarx	√						X
lwaux [4]	√				√		X
lwax [4]	√				√		X
lwbrx	√						X
lwz	√						D
lwzu	√						D
lwzux	√						X
lwzx	√						X
mcrf	√						XL
mcrfs[6]	√						X
mcrxr	√						X
mfcr	√						X
mffs*x*[6]	√						X
mfmsr [1]			√	√			X
mfspr [2]	√		√	√			XFX
mfsr [1]			√	√			X

	UISA	VEA	OEA	Supervisor Level	64-Bit	Optional	Form
mfsrin [1]			√	√			X
mftb		√					XFX
mtcrf	√						XFX
mtfsb0*x* [6]	√						X
mtfsb1*x* [6]	√						X
mtfsf*x* [6]	√						XFL
mtfsfi*x* [6]	√						X
mtmsr [1]			√	√			X
mtspr [2]	√		√	√			XFX
mtsr [1]			√	√			X
mtsrin [1]			√	√			X
mulhd*x* [4]	√				√		XO
mulhdu*x* [4]	√				√		XO
mulhw*x*	√						XO
mulhwu*x*	√						XO
mulld*x* [4]	√				√		XO
mulli	√						D
mullw*x*	√						XO
nand*x*	√						X
neg*x*	√						XO
nor*x*	√						X
or*x*	√						X
orc*x*	√						X
ori	√						D
oris	√						D
rfi [1]			√	√			XL
rldcl*x* [4]	√				√		MDS
rldcr*x* [4]	√				√		MDS
rldic*x* [4]	√				√		MD
rldicl*x* [4]	√				√		MD
rldicr*x* [4]	√				√		MD
rldimi*x* [4]	√				√		MD
rlwimi*x*	√						M
rlwinm*x*	√						M

	UISA	VEA	OEA	Supervisor Level	64-Bit	Optional	Form
rlwnm*x*	√						M
sc	√		√				SC
slbia [1,4,5]			√	√	√	√	X
slbie [1,4,5]			√	√	√	√	X
sld*x* [4]	√				√		X
slw*x*	√						X
srad*x* [4]	√				√		X
sradi*x* [4]	√				√		XS
sraw*x*	√						X
srawi*x*	√						X
srd*x* [4]	√				√		X
srw*x*	√						X
stb	√						D
stbu	√						D
stbux	√						X
stbx	√						X
std [4]	√				√		DS
stdcx. [4]	√				√		X
stdu [4]	√				√		DS
stdux [4]	√				√		X
stdx [4]	√				√		X
stfd[6]	√						D
stfdu[6]	√						D
stfdux[6]	√						X
stfdx[6]	√						X
stfiwx [5,6]	√					√	X
stfs[6]	√						D
stfsu[6]	√						D
stfsux[6]	√						X
stfsx[6]	√						X
sth	√						D
sthbrx	√						X
sthu	√						D
sthux	√						X

	UISA	VEA	OEA	Supervisor Level	64-Bit	Optional	Form
sthx	√						X
stmw [3]	√						D
stswi [3]	√						X
stswx [3]	√						X
stw	√						D
stwbrx	√						X
stwcx.	√						X
stwu	√						D
stwux	√						X
stwx	√						X
subf*x*	√						XO
subfc*x*	√						XO
subfe*x*	√						XO
subfic	√						D
subfme*x*	√						XO
subfze*x*	√						XO
sync	√						X
td [4]	√				√		X
tdi [4]	√				√		D
tlbia [1,5]			√	√		√	X
tlbie [1,5]			√	√		√	X
tlbsync [1,5]			√	√			X
tw	√						X
twi	√						D
xor*x*	√						X
xori	√						D
xoris	√						D

[1] Supervisor-level instruction
[2] Supervisor- and user-level instruction
[3] Load and store string or multiple instruction
[4] 64-bit instruction
[5] Optional in the PowerPC architecture
[6] Floating-point instructions are not supported by the MPC855T.

Appendix E
Serial ATM Scrambling, Reception, and SI Programming

E.1 ATM Cell Payload Scrambling

The ATM controller provides a scrambling option on a per line basis for cell payload bytes using the polynomial $X^{43}+1$. Figure E-1 shows the payload transmitter and receiver scrambling mechanism.

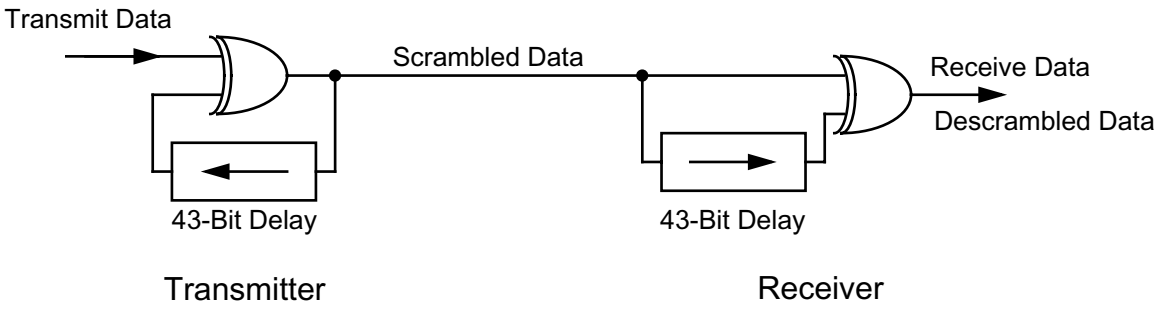


Figure E-1. ATM Cell Payload Scrambling Mechanism

The context of the transmit and receive scrambling mechanism is independent of the cell's virtual connection. The seed value for the scrambling algorithm for a particular cell is obtained from the 43 cell payload bits transmitted or received immediately prior to the current cell. Even though two consecutive cells may be from different connections, the scrambling mechanism ignores this.

E.2 Receiving Serial ATM Cells

The following figure shows the serial ATM receive procedure. After start-up when MRBLR is zero, the HEC delineation procedure begins (see below). Once complete, cell reception commences.

For each cell received, the HEC is checked. If the HEC is incorrect, the cell is still received unless the receiver loses cell delineation. If the cell is an empty cell, the payload is discarded. If there is a header match using the lookup table then the cell payload is received. When the external CAM option is enabled, the cell is received before the CAM match is checked.

Operation continues in this manner until the RESTART RECEIVE (ATM Opcode = 101) command or a FIFO overrun error occurs or the user disables the SCC receiver. Received cells may have their payloads descrambled as described earlier.

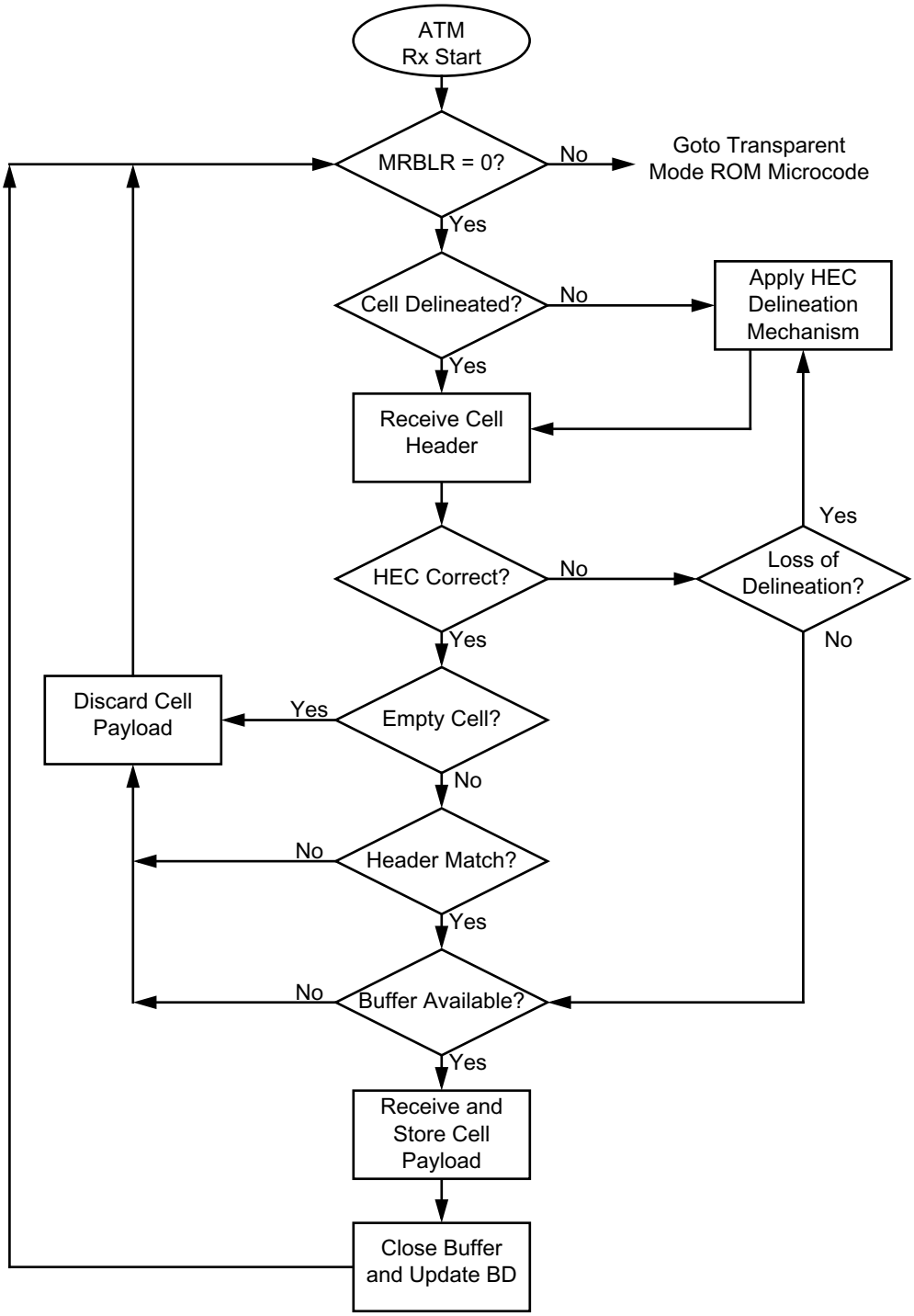


Figure E-2. Serial ATM Receive Procedure

E.2.1 HEC Delineation Mechanism

The ATM controller applies the cell delineation mechanism specified in I.432 and shown in the following figure to synchronize to the incoming cell stream. The SDH-based physical layer values for alpha and delta, 7 and 6 respectively, are used by the ATM controller.

At start-up, the hunt mode is entered where the CRC-8 value is calculated on each incoming longword. When this matches the next received byte, it is assumed that the HEC is found and the state machine advances to the presync state. Because the hunt state only finds a HEC field that is aligned to the start of reception, it can take up to four correct cells to leave the hunt state.

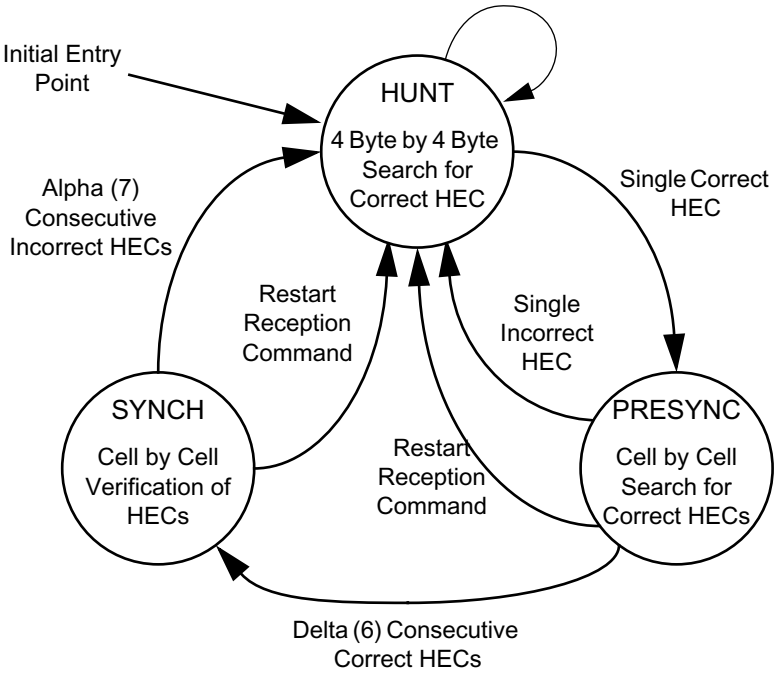


Figure E-3. Cell Delineation State Diagram

In the presync state, the ATM controller has detected cell boundaries and is verifying that they are correct. The ATM controller leaves the presync state after a single cell with an incorrect HEC is received or the RESTART RECEIVE command is given. When six consecutive cells with valid HECs are received, the ATM controller jumps to the sync state and begins reception. The state machine advances immediately after reception of the sixth correct HEC and is not delayed until the end of the cell. This means that five cells with valid HECs are discarded and the sixth is received into a data buffer.

In the synch state, the ATM controller receives data and control cells and discards incoming empty cells. When a stream of cells with incorrect HECs are received, the first six consecutive cells are received and on reception of the seventh incorrect HEC, the ATM controller immediately jumps to the hunt state and does not receive the cell with the seventh

incorrect HEC. The ATM controller also jumps to the hunt state after the RESTART RECEIVE command.

When cell payload scrambling is enabled, the descrambler will self-synchronize with the incoming data after 43 bits. Hence, after starting the receiver, the descrambler will attain synchronization during reception of the first cell in the presync state.

E.3 Serial Interface Programming Example for Serial ATM

This section provides an example of programming the SI registers for connection to a line interface device with common transmit and receive clocks.

Note: the timeslot assigner TSA must not be enabled in SIGMR until the SI RAM and other registers have been configured.

Table E-1. Serial Interface Register Programming Example for Serial ATM

Register	User Writes (hex)
SIMODE Serial Interface Mode Register	0000 0058
SIGMR Serial Interface Global Mode Register	04
SICR Serial Interface Clock Route Register	0000 0040

E.3.1 Serial Interface RAM

An example of programming the TSA with independent receiver and transmitter routing RAM tables for back-to-back ATM cell transmission and reception is given in the following table. TDM A is used and data is routed to SCC1. Other SCCs and TDM B may be used. For debug purposes, all four SI strobes are asserted during various timeslots: L1ST1 during cell header transmission, L1ST2 on HEC transmission, L1ST3 during cell header reception and L1ST4 on HEC reception.

Table E-2. ATM Cell Transmission and Reception Programming Example

SI RAM Entry	User Writes (hex)	Strobe Asserted	Remarks
Rx A 0	104E	L1ST3	Receive four bytes of cell header
Rx A 1	2042	L1ST4	Receive one byte HEC field
Rx A 2	007E	-	Receive first sixteen bytes of cell payload
Rx A 3	007E	-	Receive second sixteen bytes of cell payload
Rx A 4	007F	-	Receive final sixteen bytes of cell payload
Tx A 0	044E	L1ST1	Transmit four bytes of cell header
Tx A 1	0842	L1ST2	Transmit one byte HEC field
Tx A 2	007E	-	Transmit first sixteen bytes of cell payload

Table E-2. ATM Cell Transmission and Reception Programming Example

SI RAM Entry	User Writes (hex)	Strobe Asserted	Remarks
Tx A 3	007E	-	Transmit second sixteen bytes of cell payload
Tx A 4	007F	-	Transmit final sixteen bytes of cell payload

E.3.2 Parallel Port Registers

For serial ATM operation over TDMA, the parallel port pins should be set up to interface to external signals as shown in the following table. The TSA strobes and SDMA acknowledge signals are shown, and common receive and transmit clocks are used.

Table E-3. TDMA Port Pin Requirements

Signal	Pin	Direction
L1TXDA	PA9	Output
L1RXDA	PA8	Input
L1RSYNCA	PC4	Input
L1RCLKA	PA7	Input
L1ST1	PB19	Output
L1ST2	PB18	Output
L1ST3	PB17	Output
L1ST4	PB16	Output
$\overline{\text{SDACK1}}$	PC5	Output
$\overline{\text{SDACK2}}$	PC7	Output

To achieve this, the following table shows an example of how the port registers may be programmed.

Table E-4. Port Register Programming Example

Register	User Writes (hex)
PIPC PIP Configuration Register	0000
PADIR Port A Data Direction Register	00C0
PAPAR Port A Pin Assignment Register	01C0
PAODR Port A Open Drain Register	0000
PBDIR Port B Data Direction Register	0000 0000
PBPAR Port B Pin Assignment Register	0000 F000
PBODR Port B Open Drain Register	0000
PCDIR Port C Data Direction Register	0500

Table E-4. Port Register Programming Example

Register	User Writes (hex)
PCPAR Port C Pin Assignment Register	0D00
PCSO Port C Special Option Register	0000
PCINT Port C Interrupt Control Register	0000
PDDIR Port D Data Direction Register	0000
PDPAR Port D Pin Assignment Register	8000

Glossary of Terms and Abbreviations

The glossary contains an alphabetical list of terms, phrases, and abbreviations used in this book. Some of the terms and definitions included in the glossary are reprinted from *IEEE Std. 754-1985, IEEE Standard for Binary Floating-Point Arithmetic*, copyright ©1985 by the Institute of Electrical and Electronics Engineers, Inc. with the permission of the IEEE.

Note that some terms are defined in the context of how they are used in this book.

A

Architecture. A detailed specification of requirements for a processor or computer system. It does not specify details of how the processor or computer system must be implemented; instead it provides a template for a family of compatible *implementations*.

Asynchronous exception. *Exceptions* that are caused by events external to the processor's execution. In this document, the term 'asynchronous exception' is used interchangeably with the word *interrupt*.

Atomic access. A bus access that attempts to be part of a read-write operation to the same address uninterrupted by any other access to that address (the term refers to the fact that the transactions are indivisible). The PowerPC architecture implements atomic accesses through the **lwarx**/**stwcx.** (**ldarx**/**stdcx.** in 64-bit implementations) instruction pair.

Autobaud. The process of determining a serial data rate by timing the width of a single bit.

B

Big-endian. A byte-ordering method in memory where the address n of a word corresponds to the *most-significant byte*. In an addressed memory word, the bytes are ordered (left to right) 0, 1, 2, 3, with 0 being the most-significant byte. See Little-endian.

Blockage. A pipeline stall that occurs when an instruction occupies an execution unit and prevents a subsequent instruction from being dispatched.

Boundedly undefined. A characteristic of results of certain operations that are not rigidly prescribed by the PowerPC architecture. Boundedly-

undefined results for a given operation may vary among implementations, and between execution attempts in the same implementation.

Although the architecture does not prescribe the exact behavior for when results are allowed to be boundedly undefined, the results of executing instructions in contexts where results are allowed to be boundedly undefined are constrained to ones that could have been achieved by executing an arbitrary sequence of defined instructions, in valid form, starting in the state the machine was in before attempting to execute the given instruction.

Breakpoint. A programmable event that forces the core to take a breakpoint exception.

Burst. A bus transfer whose data phase consists of a sequence of transfers. For example, on a 64-bit bus, a four-beat burst can transfer four, 64-bit double words.

Bus parking. A feature that optimizes the use of the bus by allowing a device to retain bus mastership without having to rearbitrate.

C

Cache. High-speed memory component containing recently-accessed data and/or instructions (subset of main memory).

Cache coherency. An attribute in which an accurate and common view of memory is provided to all devices that share the a memory system. Caches are coherent if a processor performing a read from its cache is supplied with data corresponding to the most recent value written to memory or to another processor’s cache.

Cache flush. An operation that removes from a cache any data from a specified address range. This operation ensures that any modified data within the specified address range is written back to main memory. This operation is generated typically by a Data Cache Block Flush (**dcbf**) instruction.

Caching-inhibited. A memory update policy in which the *cache* is bypassed and the load or store is performed to or from main memory.

Cast-outs. *Cache blocks* that must be written to memory when a cache miss causes a cache block to be replaced.

Changed bit. One of two *page history bits* found in each *page table entry* (PTE). The processor sets the changed bit if any store is performed into the *page*. See also Page access history bits and Referenced bit.

Clear. To cause a bit or bit field to register a value of zero. The opposite of *set*.

Context synchronization. An operation that ensures that all instructions in execution complete past the point where they can produce an *exception*, that all instructions in execution complete in the context in which they began execution, and that all subsequent instructions are *fetched* and executed in the new context. Context synchronization may result from executing specific instructions (such as **isync** or **rfi**) or when certain events occur (such as an exception).

Copy-back. An operation in which modified data in a *cache block* is copied back to memory.

Critical-data first. An aspect of *burst* accesses that allow the requested data (typically a word or double word) in a *cache block* to be transferred first.

D

Denormalized number. A nonzero floating-point number whose *exponent* has a reserved value, usually the format's minimum, and whose explicit or implicit leading significand bit is zero.

Direct-mapped cache. A cache in which each main memory address can appear in only one location within the cache, operates more quickly when the memory request is a cache hit.

Direct-store. Interface available on implementations of PowerPC architecture only to support direct-store devices. When the T bit of a *segment descriptor* is set, the descriptor defines the region of memory that is to be used as a direct-store segment. Note that this facility is being phased out of the architecture and will not likely be supported in future devices. Therefore, software should not depend on it and new software should not use it.

E

Effective address (EA). The 32- or 64-bit address specified for a load, store, or an instruction fetch. This address is then submitted to the MMU for translation to either a *physical memory* address or an I/O address.

Exception. A condition encountered by the processor that requires special, supervisor-level processing.

Exception handler. A software routine that executes when an exception is taken. Normally, the exception handler corrects the condition that caused the exception, or performs some other meaningful task (that may include aborting the program that caused the exception). The address for each exception handler is identified by an exception vector offset defined by the architecture and a prefix selected via the MSR.

Extended opcode. A secondary opcode field generally located in instruction bits 21–30, that further defines the instruction type. All MPC8xx instructions are one word in length. The most significant 6 bits of the instruction are the *primary opcode*, identifying the type of instruction. *See also* Primary opcode.

Execution synchronization. A mechanism by which all instructions in execution are architecturally complete before beginning execution (appearing to begin execution) of the next instruction. Similar to context synchronization but doesn't force the contents of the instruction buffers to be deleted and refetched.

Exponent. In the binary representation of a floating-point number, the exponent is the component that normally signifies the integer power to which the value two is raised in determining the value of the represented number. *See also* Biased exponent.

F

Fetch. Retrieving instructions from either the cache or main memory and placing them into the instruction queue.

Fully-associative. Addressing scheme where every cache location (every byte) can have any possible address.

G

General-purpose register (GPR). Any of the 32 registers in the general-purpose register file. These registers provide the source operands and destination results for all integer data manipulation instructions. Integer load instructions move data from memory to GPRs and store instructions move data from GPRs to memory.

H

Harvard architecture. An architectural model featuring separate caches for instruction and data.

I

IEEE 754. A standard written by the Institute of Electrical and Electronics Engineers that defines operations and representations of binary floating-point arithmetic.

Illegal instructions. A class of instructions that are not implemented for a particular MPC8xx processor. These include instructions not defined by the architecture. In addition, for 32-bit implementations, instructions that are defined only for 64-bit implementations are considered to be illegal instructions. For 64-bit implementations instructions that are defined only for 32-bit implementations are considered to be illegal instructions.

Implementation. A particular processor that conforms to the PowerPC architecture, but may differ from other architecture-compliant implementations for example in design, feature set, and implementation of *optional* features.

Implementation-dependent. An aspect of a feature in a processor's design that is defined by a processor's design specifications rather than by the PowerPC architecture.

Implementation-specific. An aspect of a feature in a processor's design that is not required by the PowerPC architecture, but for which the PowerPC architecture may provide concessions to ensure that processors that implement the feature do so consistently.

Imprecise exception. A type of *synchronous exception* that is allowed not to adhere to the precise exception model (see Precise exception). The PowerPC architecture allows only floating-point exceptions to be handled imprecisely.

Internal bus. The bus connecting the core and system interface unit (SIU).

Instruction latency. The total number of clock cycles necessary to execute an instruction and make ready the results of that instruction.

Interrupt. An *asynchronous exception*. On MPC8xx processors, interrupts are a special case of exceptions. See also asynchronous exception.

L

Latency. The time an operation requires. For example, execution latency is the number of processor clocks an instruction takes to execute. Memory latency is the number of bus clocks needed to perform a memory operation.

Least-significant bit (lsb). The bit of least value in an address, register, data element, or instruction encoding.

Least-significant byte (LSB). The byte of least value in an address, register, data element, or instruction encoding.

Little-endian. A byte-ordering method in memory where the address n of a word corresponds to the *least-significant byte*. In an addressed memory word, the bytes are ordered (left to right) 3, 2, 1, 0, with 3 being the *most-significant byte*. See Big-endian.

M

Master, The name given to a bus device that has been granted control, or mastership, of the bus.

Memory access ordering. The specific order in which the processor performs load and store memory accesses and the order in which those accesses complete.

Memory controller. A unit whose primary function is to control the external bus memories and I/O devices.

Memory coherency. An aspect of caching in which it is ensured that an accurate view of memory is provided to all devices that share system memory.

Memory consistency. Refers to agreement of levels of memory with respect to a single processor and system memory (for example, on-chip cache, secondary cache, and system memory).

Memory management unit (MMU). The functional unit that is capable of translating an *effective* (logical) *address* to a physical address, providing protection mechanisms, and defining caching methods.

Microarchitecture. The hardware details of a microprocessor's design.

Mnemonic. The abbreviated name of an instruction used for coding.

Modified state. When a cache block is in the modified state, it has been modified by the processor since it was copied from memory. See MESI.

Munging. A modification performed on an *effective address* that allows it to appear to the processor that individual aligned scalars are stored as *little-endian* values, when in fact it is stored in *big-endian* order, but at different byte addresses within double words. Note that munging affects only the effective address and not the byte order.

Most-significant bit (msb). The highest-order bit in an address, registers, data element, or instruction encoding.

Most-significant byte (MSB). The highest-order byte in an address, registers, data element, or instruction encoding.

N

No-op. No-operation. A single-cycle operation that does not affect registers or generate bus activity.

O

OEA (operating environment architecture). The level of the architecture that describes the memory management model, supervisor-level registers, synchronization requirements, and the exception model. It also defines the time-base feature from a supervisor-level perspective. Implementations that conform to the PowerPC OEA also conform to the PowerPC UISA and VEA.

Optional. A feature, such as an instruction, a register, or an exception, that is defined by the PowerPC architecture but not required to be implemented.

Out-of-order. An aspect of an operation that allows it to be performed ahead of one that may have preceded it in the sequential model, for example, speculative operations. An operation is said to be performed out-of-order if, at the time that it is performed, it is not known to be required by the sequential execution model. *See* In-order.

Out-of-order execution. A technique that allows instructions to be issued and completed in an order that differs from their sequence in the instruction stream.

Overflow. An error condition that occurs during arithmetic operations when the result cannot be stored accurately in the destination register(s). For example, if two 32-bit numbers are multiplied, the result may not be representable in 32 bits.

P

Pace control. Controls the rate of the data flow between a master and slave.

Page. A region in memory. The OEA defines a page as a 4-Kbyte area of memory, aligned on a 4-Kbyte boundary.

Page fault. A page fault is a condition that occurs when the processor attempts to access a memory location that does not reside within a *page* not currently resident in *physical memory.* On MPC8xx processors, a page fault exception condition occurs when a matching, valid *page table entry* (PTE[V] = 1) cannot be located.

Physical memory. The actual memory that can be accessed through the system’s memory bus.

Pipelining. A technique that breaks operations, such as instruction processing or bus transactions, into smaller distinct stages or tenures (respectively) so that a subsequent operation can begin before the previous one has completed.

Precise exceptions. A category of exception for which the pipeline can be stopped so instructions that preceded the faulting instruction can complete, and subsequent instructions can be flushed and redispatched after exception handling has completed. *See* Imprecise exceptions.

Primary opcode. The most-significant 6 bits (bits 0–5) of the instruction encoding that identifies the type of instruction. S*ee* Secondary opcode.

Protection boundary. A boundary between *protection domains*.

Protection domain. A protection domain is a segment, a virtual page, a BAT area, or a range of unmapped effective addresses. It is defined only when the appropriate relocate bit in the MSR (IR or DR) is 1.

Q

Quad word. A group of 16 contiguous locations starting at an address divisible by 16.

R

rA. The **r**A instruction field is used to specify a GPR to be used as a source or destination.

rB. The **r**B instruction field is used to specify a GPR to be used as a source.

rD. The **r**D instruction field is used to specify a GPR to be used as a destination.

rS. The **r**S instruction field is used to specify a GPR to be used as a source.

Real address mode. An MMU mode when no address translation is performed and the *effective address* specified is the same as the physical address. The processor's MMU is operating in real address mode if its ability to perform address translation has been disabled through the MSR registers IR and/or DR bits.

Record bit. Bit 31 (or the Rc bit) in the instruction encoding. When it is set, updates the condition register (CR) to reflect the result of the operation.

Register indirect addressing. A form of addressing that specifies one GPR that contains the address for the load or store.

Register indirect with immediate index addressing. A form of addressing that specifies an immediate value to be added to the contents of a specified GPR to form the target address for the load or store.

Register indirect with index addressing. A form of addressing that specifies that the contents of two GPRs be added together to yield the target address for the load or store.

Reservation. The processor establishes a reservation on a *cache block* of memory space when it executes an **lwarx** or **ldarx** instruction to read a memory semaphore into a GPR.

Reserved field. In a register, a reserved field is one that is not assigned a function. A reserved field may be a single bit. The handling of reserved bits is *implementation-dependent*. Software is permitted to write any value to such a bit. A subsequent reading of the bit returns 0 if the value last written to the bit was 0 and returns an undefined value (0 or 1) otherwise.

RISC (reduced instruction set computing). An *architecture* characterized by fixed-length instructions with nonoverlapping functionality and by a separate set of load and store instructions that perform memory accesses.

S

SLB (segment lookaside buffer). An optional cache that holds recently-used *segment table entries*.**Scalability.** The capability of an architecture to generate *implementations* specific for a wide range of purposes, and in particular implementations of significantly greater performance and/or functionality than at present, while maintaining compatibility with current implementations.

Scan chain. The peripheral buffers of a device, linked in JTAG test mode, that are addressed in a shift-register fashion.

Segment table. A 4-Kbyte (1-page) data structure that defines the mapping between effective segments and virtual segments for a process. Segment tables are implemented on 64-bit processors only.

Segment table entry (STE). Data structures containing information used to translate *effective address* to physical address in a 64-bit implementation. STEs are implemented on 64-bit processors only.

Set (*v*). To write a nonzero value to a bit or bit field; the opposite of *clear*. The term ‘set’ may also be used to generally describe the updating of a bit or bit field.

Set (*n*). A subdivision of a *cache*. Cacheable data can be stored in a given location in any one of the sets, typically corresponding to its lower-order address bits. Because several memory locations can map to the same location, cached data is typically placed in the set whose *cache block* corresponding to that address was used least recently. *See* Set-associative.

Set-associative. Aspect of cache organization in which the cache space is divided into sections, called *sets*. The cache controller associates a particular main memory address with the contents of a particular set, or region, within the cache.

Significand. The component of a binary floating-point number that consists of an explicit or implicit leading bit to the left of its implied binary point and a fraction field to the right.

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.

Static branch prediction. Mechanism by which software (for example, compilers) can give a hint to the machine hardware about the direction a branch is likely to take.

Sticky bit. A bit that when *set* must be cleared explicitly.

Superscalar machine. A machine that can issue multiple instructions concurrently from a conventional linear instruction stream.

Supervisor mode. The privileged operation state of a processor. In supervisor mode, software, typically the operating system, can access all control registers and can access the supervisor memory space, among other privileged operations.

Synchronization. A process to ensure that operations occur strictly *in order*. *See* Context synchronization and Execution synchronization.

Synchronous exception. An *exception* that is generated by the execution of a particular instruction or instruction sequence. There are two types of synchronous exceptions, *precise* and *imprecise*.

System memory. The physical memory available to a processor.

T

Time-division multiplex (TDM). A single serial channel used by several channels taking turns.

TLB (translation lookaside buffer) A cache that holds recently-used *page table entries*.

Throughput. The measure of the number of instructions that are processed per clock cycle.

U

UISA (user instruction set architecture). The level of the architecture to which user-level software should conform. The UISA defines the base user-level instruction set, user-level registers, data types, floating-point memory conventions and exception model as seen by user programs, and the memory and programming models.

User mode. The unprivileged operating state of a processor used typically by application software. In user mode, software can only access certain control registers and can access only user memory space. No privileged operations can be performed. Also referred to as problem state.

V

VEA (virtual environment architecture). The level of the *architecture* that describes the memory model for an environment in which multiple devices can access memory, defines aspects of the cache model, defines cache control instructions, and defines the time-base facility from a user-level perspective. *Implementations* that conform to the PowerPC VEA also adhere to the UISA, but may not necessarily adhere to the OEA.

Virtual address. An intermediate address used in the translation of an *effective address* to a physical address.

Virtual memory. The address space created using the memory management facilities of the processor. Program access to virtual memory is possible only when it coincides with *physical memory.*

W

Watchpoint. An event that is reported, but does not change the timing of the machine.

Word. A 32-bit data element. Note that on other processors a word may be a different size.

Write-back. A cache memory update policy in which processor write cycles are directly written only to the cache. External memory is updated only indirectly, for example, when a modified cache block is *cast out* to make room for newer data.

Write-through. A cache memory update policy in which all processor write cycles are written to both the cache and memory.

Index

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X